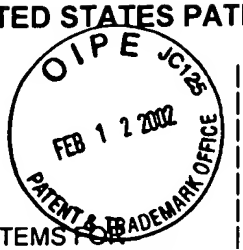
PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE



Applicant: J. Jahnke

Serial No.: 09/982,070

Filed: October 17, 2001

For: METHODS AND SYSTEMS FOR PROVIDING RESOURCES ADAPTED TO A USER ENVIRONMENT

Case No.: 30014200-1009

Group Art Unit: 2151

Examiner: not yet assigned

Date: January 16, 2002

Certificate of Mailing (37 CFR 1.8(a))
I hereby certify that this paper (along with any paper referred to as being attached or enclosed) is being deposited with the United States Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to:

Assistant Commissioner for Patents
Washington, D.C. 20231, on:

Date of Deposit: January 16, 2002

JoEllen Hogan — Date 1/16/02

SUBMISSION OF CERTIFIED COPY OF PRIORITY DOCUMENT

Assistant Commissioner for Patents
Washington, D.C. 20231

RECEIVED
FEB 19 2002
Technology Center 2100

Dear Sir:

Applicant herewith submits the certified copy of European Patent Application No. 00122627.3, filed October 17, 2000.

The Commissioner is authorized to charge any fees which may be due or credit any overpayments to Deposit Account No. 19-3140. A duplicate copy of this sheet is enclosed for that purpose.

Respectfully submitted,

SONNENSCHEIN NATH & ROSENTHAL

By: Christopher P. Rauch
Christopher P. Rauch
Registration No. 45,034

SONNENSCHEIN NATH & ROSENTHAL
P.O. Box 061080
Wacker Drive Station - Sears Tower
Chicago, Illinois 60606-1080
Telephone: (312) 876-8000

COPY OF PAPERS
ORIGINALLY FILED

Gp #2151

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: J. Jahnke

Serial No.: 09/982,070

Filed: October 17, 2001

For: METHODS AND SYSTEMS FOR PROVIDING RESOURCES ADAPTED TO A USER ENVIRONMENT

Case No.: 30014200-1009

Group Art Unit: 2151

Examiner: unknown

Date: January 16, 2002

Certificate of Mailing (37 CFR 1.8(a))
I hereby certify that this paper (along with any paper referred to as being attached or enclosed) is being deposited with the United States Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to:

Assistant Commissioner for Patents
Washington, D.C. 20231, on:

Date of Deposit: January 16, 2002

JoEllen Hogan — 1/16/02 Date

TRANSMITTAL LETTER

Assistant Commissioner for Patents
Washington, D.C. 20231

Dear Sir:

Enclosed herewith is a Submission of Certified Copy of Priority Document of J. Jahnke in the above-identified patent application entitled METHODS AND SYSTEMS FOR PROVIDING RESOURCES ADAPTED TO A USER ENVIRONMENT.

Also enclosed are: Certified Copy of European App. No. 00122627.3
Return Receipt Postcard

The Commissioner is hereby authorized to charge any additional fees required, as well as any patent application processing fees associated with this communication for which full payment has not been tendered, to Deposit Account No. 19-3140. A duplicate copy of this sheet is enclosed.

Respectfully submitted,

SONNENSCHEIN NATH & ROSENTHAL

SONNENSCHEIN NATH & ROSENTHAL
P.O. Box 061080
Wacker Drive Station - Sears Tower
Chicago, Illinois 60606-1080
Telephone: (312) 876-8000

By: Christopher P. Rauch
Christopher P. Rauch
Registration No. 45,034

Europäisches Patentamt | European Patent Office | Office européen des brevets

Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. | Patent application No. | Demande de brevet n°

00122627.3

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 10/04/01
LA HAYE, LE

Europäisches Patentamt
European Patent Office
Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.:
Application no.: 00122627.3
Demande n°:

Anmeldetag:
Date of filing: 17/10/00
Date de dépôt:

Anmelder:
Applicant(s):
Demandeur(s):
SUN MICROSYSTEMS, INC.
Palo Alto, California 94303
UNITED STATES OF AMERICA

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:
Method and apparatus for providing resources adapted to a user environment

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:
/

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE/TR
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:

82 799 q1/q4/cso

Method and Apparatus for Providing Resources Adapted to a User Environment

Field of the Invention

The present invention relates to a method and apparatus for providing resources adapted to a user environment.

Background of the Invention

In today's communication networks a user, e.g. operating a client data processing device, may access various kinds of information remotely stored at a server located at an arbitrary site throughout the world, for example stored at a data providing server connected to a network such as the Internet.

Information available for access by a client may be constituted by text information, image information, video information, audio information and similar. The information may be presented to the user, e.g., by generating a web-page at the server data processing device and transmitting display contents to the user for local display. A web page is a piece of information which may be stored at a data server and which may be accessed by a user operating a data processing device through a network. The user could for example specify an address or URL (Uniform Resource Locator) of a particular web page in the network and retrieve the corresponding information. In many cases web pages are dynamically generated upon request by the client by specific applications, for example executed at a web server in a process wherein the web page application includes text

elements, graphic elements and similar into display frames and transfers these frames to the client for local display.

Today's communication tools allow a user at any location, provided he has access to a computer network, to quickly access virtually any kind of information or services throughout the world, including for example viewing information for buying a product, retrieving published information or accessing a user account for example for setting personal data or similar.

Accordingly, a large number of users from different locations may access one and the same piece of information located somewhere and a situation may occur where users for example speaking different languages access the same information, e.g. provided on a web page, and a user may not be familiar with a language the web page is presented in and may not be able to understand the information presented. Further, information is often presented in a format specific to a country or region. Users at different locations may expect for example date, time or floating point numbers in different formats, for example in the USA a user may expect the date represented as "month/day/year" or in Germany a user may expect the date represented as "day.month.year".

Therefore information should be presentable in different languages or formats such that all potential users may access and understand the information.

As a solution it is conceivable to write, e.g. a web page application, in a number of different languages or formats, one version for each language, such that all potential users may access and understand the information presented on the web page by selecting to retrieve the web page in a desired language or format.

While this approach may work well in a case where limited amount of information is to be made available to users, in case a large amount of information, e.g. a large number of web pages or similar is provided, adapting the contents to different languages may be time consuming and cost intensive.

Summary of the Invention

It is therefore desirable to present information to different users in different versions at reduced development requirements and costs.

According to an embodiment a method provides resources to different users adapted to a user environment by setting a user parameter for selecting the user environment, executing an application independent of the user environment including reading a resource identifier also independent of the user parameter, loading a lookup object linking the resource identifier and resource data dependent on the user parameter and calling a lookup function for obtaining from the lookup object resource data based on the read resource identifier.

According thereto applications independent of a user environment may be written and executed, for example web pages independent of a language of a potential user. If a user wishes to access the application or web page, the lookup object may be used for linking, e.g. a language independent resource identifier with resource data such as a character string in a particular language or format. The retrieved resource data may then be presented to the user.

The user environment may include a user location and/or a user language preference and similar.

Resource data may be constituted by an array of at least one character and/or a resource function including rules for character representation. A resource function may include

rules for date representation and/or time representation and/or currency representation and/or floating point representation.

The lookup function may include generating a string identifier consisting of the resource identifier and the user parameter, performing the lookup function using the string identifier, and the lookup object may include an assignment between the string identifier and the resource data dependent on the user parameter.

Further the lookup function may include calling a dictionary function for obtaining one of a plurality of lookup objects corresponding to the user parameter and which may link the resource identifier with the resource data, and

Still further, the lookup object may link the resource identifier with a plurality of user parameters and each of the plurality of user parameters with resource data.

The application may be executed at an application means and may be controlled by a remotely located client data processing device, the user parameter may be received at the application means from the client, and the resource data may be transmitted to the client.

The method may further include reading resource data, and calling the lookup function for obtaining from the lookup object a resource identifier based on the read resource data.

The user parameter may be stored in an object or may be passed from a client to an application server executing the application.

A programming language may be used for implementing the method of one of the claims 1 - 11 and the programming

language may be a scripting language, e.g., the Visual Basic Scripting language.

The programming language may be the Visual Basic Scripting language and said program may be in the form of Microsoft Active Server Pages.

Further examples are disclosed in further claims.

Brief description of the drawings:

Fig. 1 shows a block diagram illustrating a system for providing resources dependent on a user environment according to an embodiment of the invention;

Fig. 2 shows a flow diagram illustrating steps of providing resources adapted to a user environment according to an embodiment of the invention,

Fig. 3a shows a flow of steps in the method according to an embodiment of the invention, particularly outlining steps for obtaining resource data,

Fig. 3b shows a flow of steps in the method according to an embodiment of the invention, particularly outlining steps for obtaining resource data,

Fig. 3c shows a flow of steps in the method according to an embodiment of the invention, particularly outlining steps for obtaining resource data,

Fig. 4 shows a block diagram illustrating elements of a system for providing resources adapted to a user environment according to another embodiment of the invention, specifically describing interaction with a client,

Fig. 5a shows a flow diagram illustrating steps of the method according to another embodiment of the invention, specifically outlining steps for providing resources to a client,

Fig. 5b shows a flow diagram illustrating steps of the method according to another embodiment of the invention, specifically outlining steps for receiving resource data from the client,

Fig. 6 shows a time sequence of processing steps of the method according to another embodiment of the invention,

Fig. 7 shows a time sequence of processing steps of the method according to another embodiment of the invention,

Fig. 8 shows a time sequence of processing steps of the method according to another embodiment of the invention, and

Fig. 9 shows a time sequence of processing steps of the method according to another embodiment of the invention.

Detailed Description of the Preferred Embodiments

In the following an embodiment of the invention will be described with respect to Fig. 1. Fig. 1 shows elements of a system for providing resources adapted to a user environment according to an embodiment of the invention. The embodiment of Fig. 1 allows to write an application independent of a user environment and allows, e.g., to provide localization to the information to be presented to a user.

Fig. 1 shows resource means 10, e.g. constituted by a data processing device such as a server having large capacity for serving a plurality of users. The resource means 10 includes user parameter means 11 for setting and/or storing a user

parameter for selecting a user environment, for example a location or a user language preference upon user input. Further, the resource means 10 includes application means 12 for executing an application independent of the user parameter including reading a resource identifier independent of the user parameter, e.g. during generating a web page. The resource means 10 further comprises lookup means 13 for loading a lookup object which links the resource identifier and resource data, the resource data being dependent on the user parameter, e.g. data to be presented to a user. Further, the lookup means calls a lookup function for obtaining from the lookup object resource data based on the read resource identifier.

Further, Fig. 1 shows a lookup table 20 preferably containing information to be accessed by the lookup object and linking the resource identifier independent of the user parameter and resource data dependent on the user parameter. The lookup table may for example be stored in a memory of the resource means or in a memory connectable to the resource means, such as a database.

The system shown in Fig. 1 allows to provide resources such as text information depending on a user environment such as a selected language and/or presentation format. An application, e.g. for providing web pages to a user may thus be written independent of the user parameter and when executed, e.g. under control of a particular user, the application and the lookup function may present information adapted to the selected user parameter, e.g. in a particular language or format which may be understood by the user.

In the following, the elements shown in Fig. 1 will be outlined in further detail.

The resource means 10 may for example be a server having large capacity connected to a computer network, such as the

Internet, and may be accessible by a plurality of users, e.g. users of different nationality, location, language and similar. The resource means 10 may also be locally connected in a local area network such as a company-wide Intranet, and may be accessible from the outside through a wide area network. The resource means 10 may further be constituted by at least one code section executed on a data processing device providing further services, e.g. network services as provided by a web server. Communication between the resource means 10 and the plurality of users may be accomplished via packet switched transmission in a network, via dedicated communication links including wireless communication.

The user parameter means 11 may be constituted by a dedicated data processing device or may be constituted by a code section containing instructions for setting at least one user parameter for selecting a user environment for the users, and may communicate with the resource means 10 via internal connections of the resource means, a network, a dedicated communication link including wireless transmission and similar.

The user parameter may be associated with a selected language, a user location or similar. For example, the user parameter may be set in accordance with a telephone country code used by a user for accessing the resource means 10, e.g. 49 for Germany and 01 for the US, or similar. In case of a user parameter indicating a location of the user in Germany German may be determined as a preferred language, in case the country code indicates the United States as user location, English could be determined as the preferred language. Further a particular format, e.g. of a current date, could be selected in accordance with the user parameter.

The user parameter means 11 may be accessed by the plurality of users for setting a user parameter for each user. This may be accomplished by requesting a user for selecting a user

parameter or the user may transmit a selected user parameter to the user parameter means 11, e.g., with a URL. A user parameter for a particular user may be set once upon starting a communication session between the resource means 10 and the user or may be set each time, e.g., in case the user transmits a request to the resource means.

It is also possible that the user is contacted by the user parameter means questioning the user to select one of a plurality of presented languages for setting the user parameter. For example, the user could be provided with a list of available languages, for example English, German, Spanish, French and Italian and could select one of these languages as preferred language. The user could also be provided with a list of available countries and/or regions, for example Germany, Spain, North America, and could select one of these countries or regions.

The selected user parameters may be stored in association with user identifiers, e.g. as an object, in a memory accessible by the user parameter means 11. Thus, the user parameter means 11 may maintain a list of user identifiers, each associated with a preferred language. In case no language is selected or the selected language is not available, a default language could be used. In case a request for an execution of an application is received at the resource means 10 from a particular user, the user parameter means in a lookup operation could determine the preferred language for that user and resources may be provided to the user in the selected language.

The resource means 10 further includes application means 12 which may be constituted by a data processing device or by a code section containing instructions for executing an application independent of a user environment, i.e., user parameter. The application means 12 could be arranged to execute application programs controlled by a user, as for

example an application program to generate a web page for user display. This could involve generating frames for display including graphic elements, text elements and similar. The application means 12 may also be arranged to independently execute applications for a plurality of users, in which case each user could control the execution of an application at the application means 12.

The applications executed at the application means 12 are independent of a selected user parameter. For example, an application controlled by a user may, instead of containing a text element in a particular language, contain a resource identifier identifying a piece of information to be included at a particular location, e.g. on a web page for display at the user. The resource identifier is independent of the user and may be translated into a particular language or format to be understood by the user in accordance with a selected user parameter. Thus, the application means may read a resource identifier independent of a user parameter and may initialize a conversion of the resource identifier into resource data in accordance with the selected user parameter for presentation to the user.

The conversion of the resource identifier into resource data for presentation to the user may be performed by the lookup means 13 arranged for calling a lookup function for obtaining from a lookup object resource data based on the read resource identifier. The lookup object preferably links a plurality of resource identifiers and resource data in dependence on a selected user parameter. The lookup means may be constituted by a data processing device or may be constituted by a code section containing instructions for performing the above outlined function. In case the lookup means is constituted by a data processing device, it may communicate with the resource means via a network or a communication link as outlined above.

Before the lookup means may access the lookup object for retrieving resource data based on a resource identifier and a user parameter, the lookup object is preferably initialized to be accessible by the lookup means. The initialization of the lookup object may be executed once upon establishing a application session with a particular user or may be initialized for each user request.

It is noted that a plurality of lookup objects may be available, and may be loaded, e.g. in dependence on requirements of an application or similar.

The lookup means 13 may receive from the application means a read resource identifier identifying a resource to be presented to a user and may call a lookup function for obtaining from the lookup object resource data based on the received resource identifier.

The resource data to be presented to the user may be constituted for example by an array of at least one character, e.g. a word in a particular language in accordance with the selected user parameter. For example, a resource identifier could identify a resource text element "intellectual property" and the resource data presented to a user for a user in Germany may for example be "Gewerblicher Rechtschutz" and for a user in France "Propriété Intellectuelle", and for a user in the United States "Intellectual Property". Of course, the selection may also be based on a language or location preference set by a user via the user parameter.

Further, the resource data may include a resource function including rules for character representation, e.g. for facilitating presentation to a user of at least one of the group consisting of date, time, currency and floating point numbers.

For example, in case the current date is to be presented to a user, the application means could call a function for obtaining the current date and could then use a function for converting the current date into a format comfortable for a user in dependence on the selected user parameter. For example, the current date "23rd of August, 2000" could be converted into "08/23/00" for a user located in the United States or making a corresponding user parameter selection. Further, for a user in Germany or a user making a corresponding selection, the current date could be converted into "23.08.00" using the resource function in dependence on the selected user parameter.

In case of a floating point number, a similar sequence of steps could be carried out. For example in case the number "1.1" should be presented in a US format, e.g. for a user in the United States making a appropriate user parameter selection, the number could be presented to the user in the form of "1.1". For a user, for example in Germany, the information could be presented as "1,1". Similar conversions of resource identifiers into resource data could be performed in other cases, for example for currency representation.

Accordingly, the embodiment described above allows to write an application independent of a user environment, i.e. of a selected user parameter, and allows to localize information to be presented to a user into a local format or language comfortable for the user. The application means may therefore execute an application which is independent of a user environment and may call a lookup function for inserting localized elements, for example into a frame containing information to be presented to a particular user.

Furthermore, it is possible that locally different user input like dates and time formats as outlined above may be converted into a format independent of a user parameter or environment, e.g. using the resource functions outlined

above. For example, an input by a user located in Germany, i.e. with a corresponding user parameter selection, of the floating point number "1,1" could be converted using an appropriate resource function into an internal format independent of the user parameter "1.1".

Still further, a user may also select multiple user parameters, e.g., a user parameter specifying language/format, as described above, and a user parameter specifying user settings, e.g. characteristics of the display of a device operated by the user, such as a personal data assistant (PDA) a palm top computer, mobile phone etc. A user parameter may also specify a user preference for the presentation of web pages, e.g. to generate customized web pages.

For example, a user may wish to view the current time and date at a particular position on the display of a data processing device or may wish a specific representation of a web page, e.g., to fit the information into a small screen of a mobile device.

To simultaneously support user parameters specifying languages/formats and user settings, a plurality of lookup objects may be employed.

In brief, the described embodiment provides internationalization support to applications which are written to be independent of a language or format desired by a particular user. Functions are provided supporting provision of localized text resources, functions which convert locally different user input like date and time and floating point representation into resource identifiers intended to store such data, and functions which convert resource identifiers such as a current date using resource functions into a localized format.

All communications between the above described elements may use internal connections of the resource means, a network, a dedicated communication link including wireless transmission and similar.

It is noted that a computer readable medium may be provided having a program recorded thereon, where the program is to make a computer or system of data processing devices execute functions of the above described elements. A computer readable medium can be a magnetic or optical or other tangible medium on which a program is recorded, but can also be a signal, e.g. analog or digital, electromagnetic or optical, in which the program is embodied for transmission.

Further, a computer program product may be provided comprising the computer readable medium.

In the following a further embodiment of the invention will be described with respect to Fig. 2. Fig. 2 shows a sequence of steps performed in the method according to another embodiment of the invention. The apparatus of the embodiment described with respect to Fig. 1 may be used for executing the sequence of steps of Fig. 2, however, Fig. 2 is not limited thereto.

Similar to the previous embodiment, the embodiment of Fig. 2 provides internationalization support to applications which are written to be independent of a language or format desired by a particular user. Functions are provided supporting provision of localized text resources, functions which convert locally different user input such as a current date into internal formats and functions which convert internal formats into a localized format.

In a first step S21 a user parameter is set for selecting a user environment. The user parameter may be received from a user and may be stored, e.g. at the user parameter means 11

outlined with respect to Fig. 1. The user parameter may correspond to a user environment including a location and/or a user language preference.

In a step S22 an application independent of the selected user parameter is executed, e.g. at the application means 12 outlined with respect to Fig. 1. The application may be controlled by the user, for example for preparing frames for display at the user including graphic elements, text elements and similar.

In a step S23 the application means reads a resource identifier independent of the user parameter. For example, the application executed at the application means may be a program in the form of Microsoft Active Server Pages (ASP) for presenting information to a user.

In a step S24 a lookup object linking the resource identifier independent of the user parameter, e.g. an identifier in an internal format, with resource data dependent on the user parameter is loaded or initialized. Step S24 may be executed once upon entering a session with a user or may be initialized in case a request is received from a user or a user parameter is set by the user. The resource data may be constituted by an array of at least one character, for example of an expression in a particular language. Further, the resource data may correspond to a resource function including rules for character representation for example for date representation and/or time representation and/or currency representation and/or floating point representation, as it was outlined with respect to the environment of Fig. 1.

Thereafter, in a step S25 a lookup function is called for obtaining from the lookup object resource data based on the read resource identifier. The obtained resource data may be presented to the user, for example an expression could be presented to a user in a language which is understood by the

user, depending on the selected user parameter, or a date or floating point representation could be presented to the user in a format the user is comfortable with. Of course, a plurality of lookup operations could be executed, in case an application includes a plurality of resource identifiers. The function for initializing or loading the lookup object and further functions as described with respect to Fig. 2 such as the lookup operation may be written in a scripting language, and the scripting language may be the Visual Basic scripting language.

Further, the steps outlined with respect to Fig. 2 may not necessarily be performed in the shown order, instead the order of the steps may be varied as appropriate. Particularly step S24 may be executed at any appropriate point in time during the operation or may be executed only once at the beginning of a session.

Therefore, after a user parameter setting a selected user environment has been obtained, the lookup object including text resources and resource functions required for the application are loaded. The lookup object loaded for the application may include specific expressions, signs and similar which need to be presented in connection with a particular application in different languages or formats. However, it is also possible that a single dictionary, e.g. lookup object, is used for all applications containing all resource data which need to be presented to a user.

After storing the dictionary, e.g. in a session object, e.g. identified by an object identifier and/or language identifier, the application, e.g. written in the form of the Microsoft Active Server Pages format can use a function which takes a resource identifier as parameter and returns the resource text, i.e. the lookup function.

The lookup function may further check whether a dictionary exists for the selected user parameter and may retrieve the text or resource function from the lookup object. In case a resource ID does not exist, an error message may be presented to the user indicating that the resource was not found.

Further, in case a resource function is used, user specific input such as date and time as well as floating point numbers may be converted from a locally dependent format into a locally independent object, e.g. a VBScript object. This object independent of the selected user parameter may then be used for processing by the application, for example in the case of floating point numbers, mathematical operations. A result of these operations may then be presented to a user as resource data in a format or form understandable by the user, i.e. dependent on the user parameter.

In case a number of N users is accessing the system, a lookup object, e.g., including text resources and resource functions may be stored for each user, e.g. in a session object, in which case the resource data are stored N times, for each user in a particular language and/or format. In this case, after a user exits the system, the lookup object may be erased. It is understood that also a set of lookup objects may be loaded for each user.

Further, it possible that for N users with M languages one lookup object is stored for all N users. In this case resource data are stored M times, i.e., in the languages/formats chosen by the users. It is understood that also a set of lookup objects may be loaded for the plurality of users. In case a large number of users accesses the system, i.e. in case N > M, this alternative efficiently uses available storage space.

Thus, the embodiment of Fig. 2 provides internationalization support to applications which are written to be independent

of a language or format desired by a particular user. Functions are provided supporting provision of localized text resources, functions which convert locally different user input like date and time and floating point representation into resource identifiers intended to store such data, and functions which convert resource identifiers such as a current date using resource functions into a localized format.

In the following a further embodiment of the invention will be described with respect to Fig. 3a. Fig. 3a shows a sequence of processing steps of the method according to another embodiment of the invention. The steps shown in Fig. 3a particularly further outline the steps for performing the lookup operation for retrieving resource data based on a resource identifier for one or a plurality of users.

The steps outlined with respect to Fig. 3a may be performed using the apparatus outlined with respect to the embodiment of Fig. 1, however, Fig. 3a is not limited thereto.

Since the steps for setting a user parameter and executing an application and reading a resource identifier are similar to the ones described with respect to Fig. 2, description starts at an entry point S24, e.g. after step S24 described with respect to Fig. 2.

In a step S3a1 a string identifier is generated, e.g. by the lookup means or a corresponding code section. The string identifier consists of the read resource identifier and the user parameter. The string identifier may be generated in a calculation operation combining the resource identifier and the user parameter.

In a step S3a2 a lookup operation is then performed by the lookup means using the user parameter and a lookup object. The lookup object in the present embodiment includes an

assignment between the string identifier and the resource data depending on the resource identifier. Accordingly, the resource identifier is first converted into the string identifier using the user parameter and then in the lookup operation corresponding resource data are retrieved, e.g. from the lookup table 20 described with respect to Fig. 1.

The read resource data depending on the user parameter may then be provided to the user.

As an example it is assumed that a resource identifier, e.g. resID_IP, being associated with the expression "Intellectual Property" is read by the application. Further, it is assumed that a user parameter 49 standing for the environment "Germany" and a user parameter 01 standing for the environment "United States" is provided.

Accordingly, in case the user parameter 49 is selected by a user, the generated string identifier consists of the resource identifier resID_IP and the user parameter 49 and may read, for example, as resID_IP_49. In the lookup object this string identifier is preferably stored in association with the German expression "Gewerblicher Rechtschutz" and, in case the lookup function for obtaining from the lookup object is called based on the user parameter 49 and the resource identifier resID_IP, the resource data "Gewerblicher Rechtschutz" will be retrieved, e.g. for presentation to the user.

In case the user parameter 01 is selected the lookup means will correspondingly generate a string identifier consisting of the resource identifier resID_IP and the user parameter 01, e.g. represented by resID_IP_01. Corresponding to the German version, the lookup object stores this string identifier resID_IP_01 in association with the English expression "Intellectual Property", which will be retrieved

by the lookup function, e.g. for presentation to the user selecting the user parameter 01.

Accordingly, in the described example the lookup object may contain the information shown in Table 1.

resID_IP_49	"Gewerblicher Rechtschutz"
resID_IP_01	"Intellectual Property"

Table 1

In case of a resource function for presenting, e.g. a floating point number to a user, the following example may occur.

It is assumed that the floating point number 1fpt7 is to be presented to a user, wherein fpt stands for floating point. Accordingly, the number to be presented to the user represents 1.7. Further, it is again assumed that a user parameter 49 stands for the user environment "Germany" and a user parameter 01 stands for a user environment "United States".

In case the user parameter 49 is selected, the string identifier will be constituted by the resource identifier fpt and the user identifier 49, e.g. yielding fpt_49. The lookup object thus preferably maintains a rule for placing a "," in a floating point number in association with the string identifier fpt_49. Accordingly, upon retrieving the rule stored in association with the string identifier fpt_49, the number 1fpt7 to be presented to a user selecting user parameter 49 will be "1,7".

Similarly, in case the selected user parameter is 01, the string identifier will be consisting of the resource parameter fpt and the user parameter 01, for example yielding fpt_01. This string identifier will be stored by the lookup

object in association with a rule for placing a "." in a floating point number. Accordingly, the floating point number 1fpt7 to be presented to a user will be converted into the form "1.7".

The following contents shown in Table 2 may be stored in the lookup object in connection with the currently described example.

fpt_49	rule for floating point ","
fpt_01	rule for floating point "."

Table 2

The described example for floating point representation may also be used in the other direction, i.e. receiving a user input and converting the user input into an internal representation for handling at the application, the internal representation being independent of user parameter. In this case the same resource function may be used in order to convert the German version representation of "1,7" into 1fpt7 and the US version "1.7" into the internal version 1fpt7.

Similar operations may be performed in connection with presenting a date or current time to a user which selected a specific user parameter.

First, the application upon detecting an instruction to present the current date to a user could call a function for obtaining the date in an internal format independent of the user parameter and could then proceed to convert the internal representation of the local date using the selected user parameter into a corresponding localized representation of the date using a resource function performing a corresponding conversion.

The embodiment described with respect to Fig. 3a allows to efficiently present a user with localized information by converting a user parameter independent resource identifier into a string identifier comprising the resource identifier and the user parameter and performing a lookup operation in a lookup object using the string identifier for retrieving localized resource data.

In the following a further embodiment of the invention will be described with respect to Fig. 3b. Fig. 3b shows a sequence of steps performed in the method according to another embodiment of the invention. Similar to the embodiment described with respect to Fig. 3a the embodiment of Fig. 3b shows an alternative of steps performed for retrieving resource data for presentation to one or a plurality of users.

Fig. 3b particularly illustrates an example wherein first one of a plurality of lookup objects is retrieved, in correspondence to a selected user parameter. Thus, for each user parameter a lookup object is available, storing resource identifiers in association with resource data. As the steps for setting a user parameter and executing an application and reading a resource identifier are similar to the steps described with respect to Fig. 2, the flow starts at an entry point S23, e.g. following step S23 of Fig. 2.

In a step S3b1 a dictionary function is called for obtaining one of a plurality of lookup objects corresponding to the user parameter, wherein the retrieved lookup object links the resource identifier with the resource data dependent on the selected user parameter. The dictionary function thus selects at least one of a plurality of lookup objects in dependence on the selected user parameter.

The dictionary function may for example be executed at the lookup means 12 outlined with respect to Fig. 1 or may be

constituted by a code section containing corresponding instructions executed at any other location. The dictionary function may receive the selected user parameter and may accordingly select at least one of a plurality of lookup objects in correspondence to the selected user parameter. The lookup object may be loaded into the environment of the application such that the lookup function may access the identified lookup object for obtaining resource data.

Thereafter, in a step S3b2 the lookup operation may be performed using the resource identifier for obtaining the resource data, e.g. for presentation to a user.

For example, in case a resource identifier for the resource "Intellectual Property" is provided, e.g. constituted by res_ID_IP, as outlined before, and in case a user parameter 49 for the environment "Germany" is provided, the lookup object may contain information as shown in Table 3.

res_ID_IP	"Gewerblicher Rechtschutz"

Table 3: Lookup object for user parameter 49 (Germany)

In case a user parameter 01 for the environment "United States" is selected, a lookup object may contain information as outlined in Table 4.

res_ID_IP	"Intellectual Property"

Table 4: Lookup object for user parameter 01 (United States)

Of course, the above-described lookup objects contain further resource identifiers and resource data.

Further, for each language identifier a plurality of lookup objects may be provided, e.g. depending on applications, requirements or similar.

The embodiment described with respect to Fig. 3b allows in a two-step operation to retrieve the required resource data, by first identifying a lookup object corresponding to the selected user identifier, e.g. selected language, and then by retrieving the resource data using the resource identifier. Accordingly, an application independent of a user environment may be executed and localized information may be introduced into, e.g. frames for presentation to a user.

In the following a further embodiment of the invention will be described with respect to Fig. 3c. Fig. 3c shows a step performed in the method according to another embodiment of the invention. As the steps for setting a user parameter, executing an application and reading a resource identifier are similar to the ones described with respect to Fig. 2, Fig. 3c starts at an entry point S24, e.g. step S24 described with respect to Fig. 2. Similar to the embodiments described with respect to Fig. 3a and Fig. 3b the embodiment of Fig. 3c particularly describes a step for obtaining resource data based on a resource identifier for one or a plurality of users.

In the present embodiment in a step S3c1 the lookup operation is executed based on a resource identifier in a lookup object, wherein the lookup object links the resource identifier with a plurality of user parameters and each of the plurality of user parameters with resource data dependent on the respective user parameter. First the resource identifier may be located in the lookup object and then the selected user parameter and thus resource data corresponding to a user identifier and a selected user parameter may be retrieved.

In the examples described with respect to Figs. 3a and 3b, in case a resource identifier res_ID_IP for "Intellectual Property" and user parameters 49 and 01 for the environments

"Germany" and "United States" are provided, the lookup object may contain information as shown in Table 5.

res_ID_IP	49	"Gewerblicher Rechtschutz"
	01	"Intellectual Property"

Table 5

Of course, the lookup object may contain a plurality of resource identifiers or resource functions as outlined with respect to previous embodiments.

In the following a further embodiment of the invention will be described with respect to Fig. 4. Fig. 4 shows elements of an apparatus according to another embodiment of the invention. Further to the elements shown in Fig. 1, Fig. 4 shows a client 40 communicating with the resource means 10. Even though only one client is shown, a plurality of clients at different locations, e.g. different countries, regions and similar may be provided.

The client 40 may be a data processing device operated by a user and may communicate with the resource means 10 via a communication network or dedicated communication link including wireless transmissions as indicated by arrows 401 and 402.

The client may control an application executed at the resource means 10 by transmitting instructions or requests to the resource means 10 as indicated by arrow 401. A request may be a request for a particular web page, e.g. using a URL (Uniform Resource Identifier) as common in network applications.

Further, the client may transmit a selected user parameter to the resource means 10, e.g. upon request by the resource

means 10, or may transmit the user parameter in association with a URL transmitted in a request to the resource means 10. Thus, the client may comprise means for selecting the user parameter at the client, e.g. by user input, and to transmit the user parameter to the data processing device, for example in a URL.

The resource means may store the user parameter transmitted from the client in an object such that the user parameter is readily available in case an application independent of the user parameter is executed at the resource means 10 and "localized" resource data need to be presented to the user.

The resource means may maintain a plurality of user parameters in the object, preferably in association with client identifiers, in order to be able to serve a large number of clients.

The resource means, upon retrieving resource data, e.g. as outlined with respect to the embodiment of Fig. 1, may transmit the resource data to the client 40 as indicated by an arrow 402.

Apart from the above-outlined modifications, the system shown in Fig. 4 is similar to the system of Fig. 1.

All communications in the shown system may be executed via communication networks, via dedicated communication links including wireless transmission or similar.

In the following a further embodiment of the invention will be described with respect to Fig. 5a. Fig. 5a shows a sequence of steps of the method according to another embodiment of the invention. The sequence of steps shown in Fig. 5a may for example be executed by the system shown in Fig. 4, however, Fig. 5a is not limited thereto. Fig. 5a particularly outlines steps for presenting "localized"

resource data to a user in accordance with a selected user environment.

In a step S5a1 a user parameter for setting a user environment is received at a resource means, e.g. resource means 10 of Fig. 4, from a client, e.g. client 40 of Fig. 4.

In a step S5a2 an application controlled by the client is executed, e.g. by application means 12 shown in Fig. 4. Further, a resource identifier independent of the user parameter is read in connection with the application.

In a step S5a3 resource data based on the resource identifier are obtained, e.g. as outlined with respect to previous embodiments.

In a step S5a4 it is decided whether the retrieved resource data correspond to a resource function and in case in step S5a4 the decision is "YES", i.e. a resource function is retrieved, the resource function is executed, e.g. in dependence on the resource identifier in a step S5a5, e.g. a format conversion of date, time, currency and floating point, as outlined with respect to previous embodiments and the resource data are transmitted to the client in a step S5a6.

In case in step S5a4 the decision is "NO", e.g. in case the resource data correspond to an array of at least one character, the resource data are directly transmitted to the client in step S5a6.

The embodiment outlined with respect to Fig. 5 allows to provide a user with e.g. localized text information in a language corresponding to the user parameter or with information in a format comfortable for the user, e.g. floating point representation, date representation and similar, as outlined with respect to previous embodiments.

In the following a further embodiment of the invention will be described with respect to Fig. 5b. Fig. 5b shows a sequence of steps executed in the method according to another embodiment of the invention. The steps of Fig. 5b may be executed by the system shown in Fig. 4, however, Fig. 5b is not limited thereto.

Fig. 5b particularly outlines steps for receiving resource data depending on a user parameter from a user, e.g. in a local language or format, and to convert the received resource data into a resource identifier independent of the user parameter.

In a step S5b1 a user parameter for setting a user environment is received at a resource means, e.g. resource means 10 of Fig. 4, from a client, e.g. client 40 of Fig. 4. However, it is also possible that the user parameter is already stored at the resource means, e.g. in an object, as outlined before.

In a step S5b2 an application controlled by the client is executed and resource data dependent on the user parameter are received at the resource means from the client. For example, the client could transmit a date in a local format to the resource means.

In a step S5b3 a resource identifier based on the received resource data may be retrieved, the resource identifier being independent of the user parameter. The resource identifier may be used for internal processing at the resource means 10. A processing result may thereafter be output again to a user, e.g. according to the method outlined with respect to Fig. 5a.

The method outlined with respect to Fig. 5 may use resource functions as outlined with respect to previous embodiments and facilitate converting information provided by a user in a

local format or language into a representation which is independent of the locality or language of the user.

In the following a further embodiment of the invention will be described with respect to Fig. 6. Fig. 6 shows a time sequence of processing steps of the method according to another embodiment of the invention. Fig. 6 particularly outlines steps for providing resource data, i.e. localized data to a client, similar to the embodiment described with respect to Fig. 3a. The sequence of processing steps may be executed by the system described with respect to Fig. 4, however, Fig. 6 is not limited thereto.

Fig. 6 shows processing steps involving a client, such as client 40 shown in Fig. 4, resource means such as resource means 10 shown in Fig. 4 and a lookup object. Even though the lookup object may be constituted by a code section executed at the resource means, for illustration purposes the lookup object is shown as a separate entity.

In a first step S601 a user parameter is received at the resource means from the client, specifying a user environment such as a location or a preferred language. The user parameter is stored at the resource means, e.g. in a list of user parameters associated with a client identifier.

In a step S602 at the resource means an application independent of the user parameter is executed and in connection therewith a resource identifier identifying a resource independent of the user parameter is read. Further, in step S602 the user parameter is combined with the resource identifier to obtain a string identifier, for example as outlined with respect to Fig. 3a.

In a step S603 the string identifier is transmitted from the resource means to the lookup object and at the lookup object in a step S604 corresponding resource data are retrieved

based on the received string identifier, as for example outlined with respect to Fig. 3a.

In a step S605 the resource data are then transmitted to the client for local display at the client. The resource data, as outlined with respect to previous embodiments, may represent expressions in a language corresponding to the selected user parameter.

Further, in case the resource data are associated with a resource function, as outlined with respect to previous embodiments, for example in case a current date, time or similar was retrieved and is to be presented to the user in a particular format, corresponding operations may be executed, e.g. at the resource means, as indicated by a step S606. Thereafter, in a step S607 the resource data are provided to the client, e.g. the current date or time in a format corresponding to the user parameter.

The resource data may also be transmitted directly from the lookup object to the client in step S605.

The described embodiment allows to provide a user with "localized information", e.g. expressions in a particular language corresponding to the user parameter or representations of for example date, time and similar in a format corresponding to the user parameter.

In the following a further embodiment of the invention will be described with respect to Fig. 7. Fig. 7 shows a time sequence of steps performed in accordance with another embodiment of the invention. The sequence of steps may be executed by the system shown in Fig. 4, however, Fig. 7 is not limited thereto.

Fig. 7 particularly illustrates steps for providing a client with resource data in correspondence to a selected user

parameter. Fig. 7 shows processing steps involving a client, resource means, a dictionary function and a lookup object. Even though the dictionary function and the lookup object may be constituted by code sections executed at the resource means, e.g. resource means 10 of Fig. 4, the dictionary function and the lookup object are shown as separate entities.

In a step S701 a user parameter selected at the client is transmitted to the resource means for setting a user parameter at the resource means in correspondence to the client, e.g. for storage in an object containing client identifiers in association with user parameters.

In a step S702 the user parameter is transmitted from the resource means to the dictionary function which selects one of a plurality of lookup objects in correspondence to the received user parameter in a step S703, as outlined with respect to embodiment of Fig. 3b.

In a step S704 the dictionary function returns a lookup object identifier identifying the detected lookup object corresponding to the user parameter.

In a step S705 the resource means transmits a read resource identifier independent of a user parameter to the detected lookup object using the received lookup object identifier.

In a step S706 the lookup object retrieves the corresponding resource data and transmits same to the client in a step S707. In case the resource data correspond to a resource function, steps similar to steps S605, S606 and S607 of Fig. 6 may be performed.

The embodiment of Fig. 7 allows a two-step operation for first identifying an appropriate lookup object corresponding to the user parameter and then retrieving the desired

resource data based on the resource identifier from the detected lookup object.

In the following, a further embodiment of the invention will be described with respect to Fig. 8. Fig. 8 shows a time sequence of steps of the method according to another embodiment of the invention. Fig. 8 illustrates an embodiment similar to the embodiment described with respect to Fig. 3c and shows steps involving a client, resource means and a lookup object, e.g. as outlined before.

In a step S801 a user parameter is received at the resource means from the client. The resource means in a step S802 reads a resource identifier independent of the user parameter in accordance with the execution of an application and transmits the user parameter and the resource identifier to the lookup object. The lookup object in a step S803 retrieves resource data corresponding to the user parameter and the resource identifier and transmits same to the client in a step S804. The steps performed according to the embodiment of Fig. 8 are similar to the ones described with respect to Fig. 3c.

In the following, a further embodiment of the invention will be described with respect to Fig. 9. Fig. 9 shows a time sequence of steps performed in the method according to another embodiment of the invention. The sequence of steps may be executed by the system described with respect to Fig. 4, however, the embodiment of Fig. 9 is not limited thereto. The steps illustrated in Fig. 9 particularly describe to convert resource data received from a client into an internal format for usage as the resource means.

Fig. 9 illustrates steps involving a client, such as client 40 of Fig. 4, resource means such as resource means 10 of Fig. 4 and a lookup object.

In a first step S901 a user parameter is received at the resource means from the client. Thereafter, in a second step S902 resource data, e.g. data in a local format or language are transmitted from the client to the resource means. The resource data may for example be a floating point number in a local format, as outlined before.

In a step S903 the resource means transmits the user parameter and the resource data to the lookup object which in a step S904 determines a resource identifier, e.g. a resource function to be applied to the resource data. The resource function or an identifier of the resource function is transmitted to the resource means in a step S905.

The resource means in a step S906 then converts the received resource data, e.g. a floating point number in a "local" format into a format internal to the resource means S906, which is independent of the user parameter, e.g. using a look up object.

The steps outlined with respect to Fig. 9 allow to convert a user input dependent on a user parameter, i.e. a "localized" input into a form which is independent of the user parameter and which may be used at the resource means for further processing.

It is noted that a computer readable medium may be provided having a program recorded thereon, where the program is to make a computer or system of data processing devices execute functions of the resource means and/or the client. A computer readable medium can be a magnetic or optical or other tangible medium on which a program is recorded, but can also be a signal, e.g. analog or digital, electromagnetic or optical, in which the program is embodied for transmission.

Further, a computer program product may be provided comprising the computer readable medium.

As described in the above embodiments, for example when writing web applications or web pages which may be used by users from different countries or localities, the applications may be developed independently of the intended users and may then be "localized" for the different countries or localities.

In an environment of for example Microsoft Active Server Pages and the Visual Basic scripting language thus a set of functions can be used to add "internationalization support". Instead of copying active server pages and then translating these copies pages into different formats or languages, the functions may be involved to convert the generic active server page, e.g. a server page independent of user parameters or user environments into a representation for display at a user which is dependent on a selected user parameter or environment.

The above described functions allow support for localized text resources, allow to convert locally different user input like date and time into internal objects and to store such data in these objects, and further provides functions which convert such objects into a localized format for a user.

Prior to using the functions the user parameter must be defined. The user parameter can be defined by offering the user a web page with a language selection. This information is then usually stored in a session object, however, the user parameter may be also passed to following pages in a URL.

In case of providing localized text resources, after the user has selected a language, i.e. user parameter, the application loads the text resources required for the application in a lookup object. The lookup object may map a resource description, i.e. the resource identifier, to the actual resource text, i.e. the resource data. The lookup object may

be stored in an application - or session - object and may be identified by a language identifier.

After initializing the lookup object, an active server page application may use the lookup function taking a resource identifier as parameter and returning a resource data depending on the user parameter. Further, a function may check whether a dictionary exists for the selected user parameter and may retrieve text from the lookup object.

In case of a non-existing resource identifier an error message may be generated.

In case locally different user input is to be converted into an internal format such as a VBScript object, for example date and time as well as floating point numbers may be converted from a locally dependent format to a locally independent VBScript object using a resource function as outlined before.

In case a user is to be provided with a localized representation of e.g. date and time or floating point numbers the internal format may be converted into the local format using a resource function, as also outlined before.

According to another embodiment resource means for providing resources depending on a user environment may include the following elements:

1) Resource means for providing resources depending on a user environment, including

- a code section containing instructions for obtaining a user parameter for setting the user environment,

- a code section containing instructions for executing an application independent of the user parameter including

reading a resource identifier independent of the user parameter, and

- a code section containing instructions for initializing a lookup object linking the resource identifier and resource data, the resource data being dependent on the user parameter, and for calling a lookup function for obtaining from the lookup object resource data based on the read resource identifier.

2) Resource means of 1), wherein the resource data are constituted by at least on one of the group consisting of

- an array of at least one character, and

- a resource function including rules for character representation.

3) Resource means of 1), wherein the resource function includes rules for at least one of the group consisting of

- date representation,

- time representation,

- currency representation, and

- floating point representation.

4) Resource means of 1), wherein the user environment includes at least one of the group consisting of

- a location, and

- a user language preference.

5) Resource means of 1), including a code section containing instructions for loading at least one lookup object for each user.

6) Resource means of 1), including a code section containing instructions for loading at least one lookup object for a plurality of users.

7) Resource means of 1), wherein

- the lookup means includes

  - a code section containing instructions for generating a string identifier consisting of the resource identifier and the user parameter,

  - a code section containing instructions for performing the lookup function using the string identifier, and

- the lookup object includes an assignment between the string identifier and the resource data dependent on the user parameter.

8) Resource means of 1), comprising a code section containing instructions for calling a dictionary function for obtaining one of a plurality of lookup objects corresponding to the user parameter and which links the resource identifier with the resource data.

9) Resource means of 1), wherein the lookup object links the resource identifier with a plurality of user parameters and each of the plurality of user parameters with resource data.

10) Resource means of 1), wherein

- the application executed at the application means is controlled by a client,

- a code section containing instructions is provided for receiving the user parameter from the client, and

- a code section containing instructions is provided for transmitting the resource data to the client.

11) Resource means of 1), wherein

- a code section containing instructions is provided for reading resource data, and

- a code section containing instructions is provided for obtaining from the lookup object a resource identifier based on the read resource data.

12) Resource means of 1), wherein the user parameter is stored in an object.

13) Resource means of 1), wherein the user parameter is passed from a client to an application server executing the application.

According to another embodiment a client adapted to cooperate with the resource means for providing resources depending on a user environment may include the following elements:

14) Client adapted to cooperate with the resource means of one of 1) to 13), including

- a code section containing instructions for setting the user parameter, and

- a code section containing instructions for transmitting the user parameter to the data processing device.

According to another embodiment a data processing system for providing resources depending on a user environment may include the following elements:

15) Data processing system for providing resources depending on a user environment, comprising a client and resource means, wherein

- the client includes

  - a code section containing instructions for setting a user parameter for selecting the user environment, and

  - a code section containing instructions for transmitting the user parameter to the data processing device,

- the resource means includes

  - a code section containing instructions for obtaining a user parameter for setting the user environment,

  - a code section containing instructions for executing an application independent of the user environment including reading a resource identifier independent of the user parameter, and

  - a code section containing instructions for calling a lookup function for obtaining from the lookup object resource data based on the read resource identifier and for loading a lookup object linking the resource identifier and resource data dependent on the user parameter.

In the following examples of code sections for the functions outlined above are given. It is noted that the code sections are examples for selected functions only.

CODE EXAMPLES

```
<%
 '<!-- #include virtual = "/Inc/121/Array.inc" -->
 '<!-- #include virtual = "/Inc/121/File.inc" -->
 '<!-- #include virtual = "/Inc/121/String.inc" -->
 '<!-- #include virtual = "/Inc/121/Util.inc" -->
%>


<%
'/**
' * functions for internationalisation for web pages.
' *
' */
'public class International { private International() {}
%>

<%
'/**
' * names of Session-Variables / Cookies, for storing language tables.
' */
'public final static String INT_APPLICATION_DICTIONARY, INT_SESSION_LANGUAGE,
INT_COOKIE_LANGUAGE;
public const INT_APPLICATION_DICTIONARY = "SessionStringRessourceDictionary"
public const INT_SESSION_LANGUAGE   = "SessionLanguage"
public const INT_COOKIE_LANGUAGE    = "SessionLanguage"

'public final static String INT_FORM_LANGUAGE;
public const INT_FORM_LANGUAGE      = "Int_Sprache"

'public final static String INT_SESSION_CURRENCY, INT_SESSION_DEFAULTCOUNTRY,
INT_SESSION_DATEDELIMITER;
public const INT_SESSION_CURRENCY   = "Int_Wahrungssymbol"
public const INT_SESSION_ORDERCALC  = "Int_Auftragsberechnung"
public const INT_SESSION_ACCOUNTING = "Int_Buchhaltungstyp"
public const INT_SESSION_DEFAULTCOUNTRY = "Int_DefaultLand"
public const INT_SESSION_DATEDELIMITER  = "Int_DatumsTrenner"

'// Default language is German
private const INT_DEFAULT_LANGUAGE  = "49"

Dim INT_WORD_DELIMITER
INT_WORD_DELIMITER                  = vbTab

Dim INT_LANGUAGE
INT_LANGUAGE = INT_DEFAULT_LANGUAGE

private const INT_STORAGE_APPLICATION = 1
private const INT_STORAGE_SESSION     = 2
Dim INT_LANGUAGETABLE_STORAGE
INT_LANGUAGETABLE_STORAGE = INT_STORAGE_APPLICATION


'/**
' * determine language code
' */
'    public String Int_getLanguage() {}
function Int_getLanguage()
    if Util_hasSession() then
        if String_safeCStr( Session( INT_SESSION_LANGUAGE ) ) = "" then
            Session(      INT_SESSION_LANGUAGE     )      =      Request.Cookies(
INT_COOKIE_LANGUAGE )
            if String_safeCStr( Session( INT_SESSION_LANGUAGE ) ) = "" then
                Session( INT_SESSION_LANGUAGE ) = INT_DEFAULT_LANGUAGE
            end if
        end if
        Int_getLanguage = Session( INT_SESSION_LANGUAGE )
    else
        Int_getLanguage = INT_LANGUAGE
    end if
    if Len( Int_getLanguage ) < 2 then Int_getLanguage = "0" + Int_getLanguage
end function

'/**
' * set new language
' *
' * @param  sLanguage       neuer Sprachschluessel (Laendervorwahl)
' * @author Joerg Jahnke (JJ)   21.04.98
```

```
' * @change Joerg Jahnke (JJ)    05.10.98    sofern keine Session vorhanden ist,
die Sprache in der Variablen INT_LANGUAGE speichern
' */
'    public void Int_setLanguage( String sLanguage ) {}
sub Int_setLanguage( sLanguage )
    if Int_getLanguage() <> sLanguage then
        'call Int_clearStringRes()
        if Util_hasSession() then
            Session( INT_SESSION_LANGUAGE ) = sLanguage
        else
            INT_LANGUAGE = sLanguage
        end if
    end if
end sub


'/**
' * show Listbox with available languages.
' *
' * @param  sLanguages
' * @param  bHideIfOneLanguage

' */
'        public  void  Int_showLanguageSelection( String  sLanguages,  boolean
bHideIfOneLanguage ) {}
sub Int_showLanguageSelection( sLanguages, bHideIfOneLanguage )
    ' Dictionary-Objekt mit zur Auswahl angebotenen Sprachen
    Dim aLanguageDict

    Set aLanguageDict = Server.CreateObject( "Scripting.Dictionary" )
    call aLanguageDict.Add( "49", "Deutsch" )
    call aLanguageDict.Add( "01", "English" )

    Dim aLanguageArray, sLanguage

    if String_safeCStr( sLanguages ) <> "" then
        ' Dann Sprachen aus Parameterstring auslesen
        aLanguageArray = Split( sLanguages )
    else
        ' Sonst alle Sprachen waehlen
        aLanguageArray = aLanguageDict.Keys()
    end if
    if Array_safeUBound( aLanguageArray ) = 0 and bHideIfOneLanguage then
%>
        <INPUT type=hidden name="<% = INT_FORM_LANGUAGE %>" value="<% = Trim(
sLanguages ) %>">
<%
    else
%>
        <SELECT name="<% = INT_FORM_LANGUAGE %>">
<%
        for each sLanguage in aLanguageArray
            if aLanguageDict( sLanguage ) <> "" then
%>
                <OPTION  value="<%  =  sLanguage  %>"<%  if  Request.Cookies(
INT_COOKIE_LANGUAGE  )  =  sLanguage  then  %>  selected<%  end  if  %>><%  =
aLanguageDict( sLanguage ) %></OPTION>
<%
          end if
        next
%>
        </SELECT>
<%
    end if
%>
<%
end sub


'/**
' * determine whether language table exists in application variable
' *
' * @param  sLanguage
' * @return true, if Application-Variable exists
' */
'    public boolean Int_applicationDictionaryExists( String sLanguage ) {}
function Int_applicationDictionaryExists( sLanguage )
    Dim sDictionary

    sDictionary = INT_APPLICATION_DICTIONARY & sLanguage

    Int_applicationDictionaryExists = not isEmpty( Application( sDictionary ) )
end function
```

```
'/**
' * determine whether language table exists in Session-Variable
' *
' * @param  sLanguage
' * @return true, if yes
' */
'    public boolean Int_sessionDictionaryExists( String sLanguage ) {}
function Int_sessionDictionaryExists( sLanguage )
    Dim sDictionary

    sDictionary = INT_APPLICATION_DICTIONARY & sLanguage

    Int_sessionDictionaryExists = false
    on error resume next
    Int_sessionDictionaryExists = not isEmpty( Session( sDictionary ) )
    Int_sessionDictionaryExists  =  Int_sessionDictionaryExists  and  (  Session(
sDictionary ).Count > 0 )
end function

'/**
' * determine whether language tables exist
' */
'    public boolean Int_dictionaryExists() {}
function Int_dictionaryExists()
    Int_dictionaryExists = Int_dictionaryExists2( Int_getLanguage() )
end function

'/**
' * determine object with language table.
' *
' * @param  sLanguage -
' * @return Reference to language table object
' */
'    public Object Int_getDictionary( String sLanguage ) {}
function Int_getDictionary( sLanguage )
    Dim bSessionDictionary, bApplicationDictionary, sDictionary

    sDictionary = INT_APPLICATION_DICTIONARY & sLanguage

    if not Int_dictionaryExists2( sLanguage ) then
        Set Int_getDictionary = Server.CreateObject( "Scripting.Dictionary" )
        call Err.Clear()
        on error resume next
        Set Application( sDictionary ) = Int_getDictionary
        if Err.Number <> 0 then Set Session( sDictionary ) = Int_getDictionary
    else
        if Int_applicationDictionaryExists( sLanguage ) then
            Set Int_getDictionary = Application( sDictionary )
        else
            Set Int_getDictionary = Session( sDictionary )
        end if
    end if
end function


'/**
' * append file to language table.
'     public boolean Int_loadStringRes( String sFilename, String sPath, boolean
bOverwrite ) {}
function Int_loadStringRes( sFilename, sPath, bOverwrite )
    Int_loadStringRes   =   Int_loadStringRes2(  sFilename,   sPath,   bOverwrite,
Int_getLanguage() )
end function

'/**
' * append file to language table.

' */
'     public boolean Int_loadStringRes2( String sFilename, String sPath, boolean
bOverwrite, String sLanguage ) {}
function Int_loadStringRes2( sFilename, sPath, bOverwrite, sLanguage )
    Int_loadStringRes2 = true

    Dim sFileContent, aLineArray, sLine, aWordArray, sKey, sValue, aDictionary

    '// Objekt mit Sprachtabellen ermitteln
    Set aDictionary = Int_getDictionary( sLanguage )

    '// Dateinamen die Endung entsprechend der Sprache hinzufuegen und Datei
auslesen
    sFileContent = File_readVirtual( CStr( sFilename ) & "." & sLanguage, sPath )
```

```
    call Util_assert( sFileContent <> "", "Int_loadStringRes2: No data in
language dictionary or dictionary not found: " + CStr( sFilename ) + "." +
sLanguage )
    aLineArray = Split( sFileContent, FILE_LINE_DELIMITER )
    for each sLine in aLineArray
        if sLine <> "" then
            '// Schluessel und Wert aus der Zeile ermitteln
            aWordArray = Split( sLine, INT_WORD_DELIMITER )
            '// Zeile OK ?
            call Util_assert( Array_safeUBound( aWordArray ) >= 1,
"Int_loadStringRes2: Illegal line in language dictionary: " + sFilename + "," +
sLine )
            sKey = aWordArray( 0 )
            sValue = aWordArray( 1 )
            '// Schluessel bereits vorhanden ?
            if aDictionary.Exists( sKey ) then
                '// Ja, ggf. uberschreiben
                if bOverwrite then
                    aDictionary( sKey ) = sValue
                else
                    '// Fehler ausgeben
                    call Util_assert( aDictionary( sKey ) = sValue,
"Int_loadStringRes2: Key already in language dictionary with a different value:"
+ sFilename + "," + sKey )
                end if
            else
                '// Nein, Schluessel-Wert-Paar der Tabelle hinzufuegen
                call aDictionary.Add( sKey, sValue )
            end if
        end if
    next
end function

'/**
' * return text corresponding to key.
' */
'    public void Int_write( String sKey ) {}
sub Int_write( sKey )
    call Response.Write( Int_getText( sKey ) )
end sub

'    public String Int_getText( String sKey ) {}
function Int_getText( sKey )
    Int_getText = Int_getText2( sKey, Int_getLanguage() )
end function


'    public String Int_getText2( String sKey, String sLanguage ) {}
function Int_getText2( sKey, sLanguage )
    '// keine Sprachtabelle fuer die Anwendung vorhanden => Fehler
    call Util_assert( Int_dictionaryExists2( sLanguage ), "Int_getText2: No
existing language table ! Key=" + sKey )

    '// Sprachtabelle ermitteln
    Dim aDict

    Set aDict = Int_getDictionary( sLanguage )

    '// Schluessel vorhanden ?
    if aDict.Exists( sKey ) then
        '// Wert aus Sprachtabelle auslesen
        Int_getText2 = aDict.Item( sKey )
    else
        '// Text mit Fehlermeldung ausgeben
        Int_getText2 = "Int_getText: Key not contained in language table: " +
sKey
    end if
end function


'/**
' * determine currency symbol (DM, $ etc.).
' * DM Default.
' *
' * @return string with currency symbol

' */
'    public String Int_getCurrency() {}
function Int_getCurrency()
    Int_getCurrency = Session( INT_SESSION_CURRENCY )
    if Int_getCurrency = "" then Int_getCurrency = "DM"
end function
```

```
'/**
' * determine booking type
' */
'    public String Int_getAccounting() {}
function Int_getAccounting()
    Int_getAccounting = Session( INT_SESSION_ACCOUNTING )
    if String_safeTrim( Int_getAccounting ) = "" then Int_getAccounting = 1
end function

'/**
' * determine algorithm for order calculation.
' * 1 (Netto) Default.
' *
' * @return AuftragsBerechnungsNr
' */
'    public String Int_getOrderCalc() {}
function Int_getOrderCalc()
    Int_getOrderCalc = Session( INT_SESSION_ORDERCALC )
    if String_safeTrim( Int_getOrderCalc ) = "" then Int_getOrderCalc = 1
end function

'/**
' * determine default for LandNr.
' * default 4 for Germany.
' *
' * @return aktuelle LandNr
' */
'    public String Int_getCountry() {}
function Int_getCountry()
    Int_getCountry = Session( INT_SESSION_DEFAULTCOUNTRY )
    if Int_getCountry = "" then Int_getCountry = 4
end function

'/**
' * determine Default- value for currency symbol and country.
' *
' * @param  aCon        connection to StarDivision- DB
' */
'    public String Int_setCountryDefaults( Object aCon ) {}
function Int_setCountryDefaults( aCon )
    Dim sSQLStr, aRS

    sSQLStr = "SELECT * FROM Mandant WHERE Status = 1"
    sSQLStr = "SELECT M.Waehrung, M.Land, M.BuchhaltungsTypNr, AB.Nr AS
AuftragsBerechnungsNr"
            & " FROM AuftragsBerechnung AB (NOLOCK)"
            & " INNER JOIN Mandant M (NOLOCK) ON AB.MandantNr = M.Nr" _
            & " WHERE M.Status = 1"
            & "   AND ( GueltigVon IS NULL OR GueltigVon <= GETDATE() ) AND (
GueltigBis IS NULL OR GueltigBis > GETDATE() )"
    Set aRS = aCon.Execute( sSQLStr )
    if not ( aRS.EOF or aRS.BOF ) then
        Session( INT_SESSION_CURRENCY )       = aRS( "Waehrung" )
        Session( INT_SESSION_DEFAULTCOUNTRY ) = aRS( "Land" )
        Session( INT_SESSION_ORDERCALC )      = aRS( "AuftragsBerechnungsNr" )
        Session( INT_SESSION_ACCOUNTING )     = aRS( "BuchhaltungsTypNr" )
        aRS.Close
        Int_setCountryDefaults = true
    else
        Int_setCountryDefaults = false
    end if
    Set aRS = nothing
end function


'/**
' * show amount with actual currency symbol (DM, $ etc.)
' * DM Default.
' */
'    public String Int_formatCurrency( double n ) {}
function Int_formatCurrency( n )
    Int_formatCurrency = Int_formatCurrency2( n, Int_getCurrency() )
end function

'/**
' * show amount with currency symbol (DM, $ etc.) of given currency.
' */
'    public String Int_formatCurrency2( double n, Object aCon, int nCurrency ) {}
function Int_formatCurrency2( n, sCurrency )
    on error resume next
    Int_formatCurrency2 = formatNumber( n, 2 ) & " " & sCurrency
end function
```

```
end function

'/**
' * convert floating point character string.
' */
'    public double Int_CDbl( String s ) {}
function Int_CDbl( s )
    Int_CDbl = null

    Dim sNum

    if InStr( s, "." ) >= 1 and InStr( s, "," ) >= 1 then
        sNum = Util_condExpr( InStr( s, "." ) >= InStr( s, "," ), Replace( s,
",", "" ), Replace( s, ".", "" ) )
    else
        sNum = String_safeTrim( s )
    end if
    on error resume next
    if InStr( CStr( 1.5 ), "," ) > 0 then
        Int_CDbl = CDbl( Replace( sNum, ".", "," ) )
    else
        Int_CDbl = CDbl( Replace( sNum, ",", "." ) )
    end if
end function

'/**
' * convert date character string in German or US format into date object
' */
'    public Date Int_CDate( String s ) {}
function Int_CDate( s )
    Int_CDate = null

    on error resume next
    if String_safeTrim( s ) <> "" then
        Dim aDateArray, sDate

        sDate = Split( String_safeTrim( s ) )( 0 )
        if InStr( sDate, "/" ) > 0 then
            aDateArray = Split( sDate, "/" )
            Int_CDate = DateSerial( aDateArray( 2 ), aDateArray( 0 ), aDateArray(
1 ) )
        else
            aDateArray = Split( sDate, "." )
            Int_CDate = DateSerial( aDateArray( 2 ), aDateArray( 1 ), aDateArray(
0 ) )
        end if

        if InStr( String_safeTrim( s ), ":" ) > 0 then
            Dim aTimeArray, sTime

            sTime = Split( String_safeTrim( s ) )( 1 )
            aTimeArray = Split( sTime, ":" )
            Int_CDate = Int_CDate + TimeSerial( aTimeArray( 0 ) + Util_condExpr(
InStr( s, "PM" ) > 0, 12, 0 ), aTimeArray( 1 ), aTimeArray( 2 ) )
        end if
    end if
end function
%>

<%
'}
%>


<%
 '<!-- #include virtual = "/Inc/121/International.inc" -->
%>

<%
'public class Date{ private Date() {}

'/**
' * month and maximum number of days
' */
'    public final static String DATE_MONTH_NAMES_49 = "";
'    public final static String DATE_MONTH_DAYCOUNT = "";
public const DATE_MONTH_NAMES_49 = "Januar Februar März April Mai Juni Juli
August September Oktober November Dezember"
public const DATE_MONTH_DAYCOUNT = "31 29 31 30 31 30 31 31 30 31 30 31"

'/**
```

```
'/**
' * determine exchange rate.
' *
' * @param  aCon             connection to DB
' * @param  sCurrencyFrom    source symbil
' * @param  sCurrencyTo      target symbol
' * @return multiplier for amounts
' */
function Int_getExchangeRate( aCon, sCurrencyFrom, sCurrencyTo )
    const INT_EXCHANGERATE_DELIM1       = "#$#"
    const INT_EXCHANGERATE_DELIM2       = "$#$"
    const INT_EXCHANGERATE_TIMEOUT_SEC  = 120
    const INT_EXCHANGERATE_IDX_CURR     = 0
    const INT_EXCHANGERATE_IDX_RATE     = 1
    const INT_EXCHANGERATE_IDX_DATE     = 2
    const INT_SESSION_EXCHANGERATES     = "Int_Wechselkurse"


    if sCurrencyFrom = sCurrencyTo then
        ' dann Wechselkurs = 1
        Int_getExchangeRate = 1.0
    else
        Dim sExchangeRates, sExchangeFromTo

        sExchangeRates = String_safeCStr( Session( INT_SESSION_EXCHANGERATES ) )
        sExchangeFromTo = sCurrencyFrom & "->" & sCurrencyTo
        if sExchangeRates <> "" then
            Dim aExchangeRatesArray, sExchangeRate, aExchangeRateArray

            aExchangeRatesArray = Split( sExchangeRates, INT_EXCHANGERATE_DELIM1
)
            for each sExchangeRate in aExchangeRatesArray
                aExchangeRateArray = Split( sExchangeRate,
INT_EXCHANGERATE_DELIM2 )
                ' Gewuenscht Umrechnung ?
                if aExchangeRateArray( INT_EXCHANGERATE_IDX_CURR ) =
sExchangeFromTo then
                    if DateDiff( "s", CDate( aExchangeRateArray(
INT_EXCHANGERATE_IDX_DATE ) ), Now() ) < INT_EXCHANGERATE_TIMEOUT_SEC then
                        Int_getExchangeRate = CDbl( aExchangeRateArray(
INT_EXCHANGERATE_IDX_RATE ) )
                        exit function
                    else
                        Session( INT_SESSION_EXCHANGERATES ) = Join( Filter(
aExchangeRatesArray, sExchangeFromTo, false ), INT_EXCHANGERATE_DELIM1 )
                        exit for
                    end if
                end if
            next
        end if

        Dim sSQLStr, aRS, nMultFrom, nMultTo

        sSQLStr = "SELECT Zeichen, CONVERT( real, Zaehlerkurs ) / CONVERT( real,
Nennerkurs ) AS Multiplikator"
                & " FROM Waehrung"
                & " WHERE Zeichen IN ( '" & sCurrencyFrom & "','" & sCurrencyTo &
"' )"
        Set aRS = aCon.Execute( sSQLStr )
        if aRS( "Zeichen" ) = sCurrencyFrom then nMultFrom = aRS( "Multiplikator"
)
        if aRS( "Zeichen" ) = sCurrencyTo   then nMultTo   = aRS( "Multiplikator"
)
        aRS.MoveNext
        if not ( aRS.EOF or aRS.BOF ) then
            if aRS( "Zeichen" ) = sCurrencyFrom then nMultFrom = aRS(
"Multiplikator" )
            if aRS( "Zeichen" ) = sCurrencyTo   then nMultTo   = aRS(
"Multiplikator" )
        end if
        aRS.Close
        Set aRS = nothing
        Int_getExchangeRate = nMultFrom / nMultTo

        sExchangeRate = Join( Array( sExchangeFromTo, Int_getExchangeRate, Now()
), INT_EXCHANGERATE_DELIM2 )
        if String_safeCStr( Session( INT_SESSION_EXCHANGERATES ) ) = "" then
            Session( INT_SESSION_EXCHANGERATES ) = sExchangeRate
        else
            Session( INT_SESSION_EXCHANGERATES ) = Session(
INT_SESSION_EXCHANGERATES ) & INT_EXCHANGERATE_DELIM1 & sExchangeRate
        end if
    end if
```

```
' * Konstanten aus VBScript-Doku.
' */
'//const vbSunday               = 1 ' Sunday
'//const vbMonday               = 2 ' Monday
'//const vbTuesday              = 3 ' Tuesday
'//const vbWednesday            = 4 ' Wednesday
'//const vbThursday             = 5 ' Thursday
'//const vbFriday               = 6 ' Friday
'//const vbSaturday             = 7 ' Saturday
'//const vbFirstJan1                = 1 ' Use the week in which January 1 occurs
(default).
'//const vbFirstFourDays         = 2 ' Use the first week that has at least four
days in the new year.
'//const vbFirstFullWeek        = 3 ' Use the first full week of the year.
'//const vbUseSystem                = 0 ' Use the date format contained in the
regional settings for your computer.
'//const vbUseSystemDayOfWeek   = 0 ' Use the day of the week specified in your
system settings for the first day of the week.

'//const vbGeneralDate = 0 ' Display a date and/or time. For real numbers,
display a data and time. If there is no fractional part, display only a date. If
there is no integer part, display time only. Date and time display is determined
by your system settings.
'//const vbLongDate     = 1 ' Display a date using the long date format specified
in your computer's regional settings.
'//const vbShortDate    = 2 ' Display a date using the short date format specified
in your computer's regional settings.
'//const vbLongTime     = 3 ' Display a time using the long time format specified
in your computer's regional settings.
'//const vbShortTime    = 4 ' Display a time using the short time format specified
in your computer's regional settings.


'/**
' * compare difference of day number of two dates
' *
' * @param  aDate1       first date
' * @param  aDate2       second date
' * @return number of days
' */
'    public int Date_dayDiff( Date aDate1, Date aDate2 ) {}
function Date_dayDiff( aDate1, aDate2 )
    Date_dayDiff = DateDiff( "d", aDate1, aDate2 )
end function


'/**
' * determine date and time from string.
' *
' * @param  sDate        string with date in format dd.mm.yyyy or
' *                      mm/dd/yyyy and optionally time as hh:mm:ss
' * @return date or null, if no valid date
' */
'    public Date Date_fromStringWTime( String sDate ) {}
function Date_fromStringWTime( sDate )
    Date_fromStringWTime = null

    on error resume next
    Date_fromStringWTime = Int_CDate( sDate )
    call Err.Clear()
end function

'/**
' * determine date from string.
' *
' * @param  sDate        date as dd.mm.yyyy or mm/dd/yyyy
' * @return date or null, if no valid date
' */
'    public Date Date_fromString( String sDate ) {}
function Date_fromString( sDate )
    Dim sDatePart

    on error resume next
    sDatePart = Trim( sDate )
    if InStr( sDatePart, " " ) > 0 then
        sDatePart = Split( sDatePart, " " )( 0 )
    end if
    Date_fromString = Date_fromStringWTime( sDatePart )
end function

'/**
```

```
' * convert date in string of mm/dd/yyyy.
' *
' * @param  aDate       date object
' * @return string with date
' */
'    public String Date_toString( Date aDate ) {}
function Date_toString( aDate )
    Date_toString = CStr( Month( aDate ) ) + "/" + CStr( Day( aDate ) ) + "/" +
CStr( Year( aDate ) )
end function


'/**
' * date of Easter sunday.
' *
' * @param  nYear
' * @return date of Easter sunday
' */
'    public Date Date_getEasterSunday( int nYear ) {}
function Date_getEasterSunday( nYear )
    Dim a,b,c,d,e,OT,OM

    a = nYear Mod 19
    b = nYear Mod 4
    c = nYear Mod 7
    d = (19 * a + 24) Mod 30
    e = (2 * b + 4 * c + 6 * d + 5) Mod 7
    OT = 22 + d + e
    OM = 3
    If OT > 31 Then
    OT = d + e - 9
    OM = 4
    End If
    If OT = 26 And OM = 4 Then
    OT = 19
    End If
    If OT = 25 And OM = 4 And d = 28 And e = 6 And a > 10 Then
    OT = 18
    End If

    Date_getEasterSunday = DateSerial( nYear, OM, OT )
end function

'/**
' * determine whether given date is public holiday.
' *
' * @param  aDate       zu pruefendes Datum
' * @param  sBundesland state
' * @return TRUE => public holiday, FALSE => no public holiday
' */
'    public boolean Date_isPublicHoliday( Date aDate, String sBundesland ) {}
function Date_isPublicHoliday( aDate, sBundesland )
    Dim sMonth, sDay, sDate, bIsPublicHoliday, nDayDiff

    Date_isPublicHoliday = false
    sMonth = Right( "0" + CStr( Month( aDate ) ), 2 )
    sDay = Right( "0" + CStr( Day( aDate ) ), 2 )
    sDate = sDay + sMonth
    ' Neujahr, 1.Mai, Dt.Einheit, Weihnachten ?
    bIsPublicHoliday = ( sDate = "0101" ) or ( sDate = "0105" ) or ( sDate =
"0310" ) or ( sDate = "2512" ) or ( sDate = "2612" )

    if not bIsPublicHoliday then
        Dim aEasterDate

        aEasterDate = Date_getEasterSunday( Year( aDate ) )
        nDayDiff = Date_dayDiff( aEasterDate, aDate )
        bIsPublicHoliday = ( nDayDiff = 0 ) or ( nDayDiff = 1 ) or ( nDayDiff = -
2 ) or ( nDayDiff = 39 ) or ( nDayDiff = 49 ) or ( nDayDiff = 50 )
    end if
    Date_isPublicHoliday = bIsPublicHoliday
end function

'/**
' * determine whether date is a weekday.
' *
' * @param  aDate
' * @return TRUE => weekday, FALSE => Saturday or Sunday
' */
'    public boolean Date_isMondayToFriday( Date aDate ) {}
function Date_isMondayToFriday( aDate )
    Date_isMondayToFriday = ( ( Weekday( aDate ) >= vbMonday ) and ( Weekday(
aDate ) <= vbFriday ) )
```

```
end function

'/**
' * determine whether day is workday
' *
' * @param  aDate        zu pruefendes Datum
' * @return TRUE, workday FALSE not workday
' */
'    public boolean Date_isWorkDay( Date aDate ) {}
function Date_isWorkDay( aDate )
    Date_isWorkDay  =  (  Date_isMondayToFriday(  aDate  )  and  (  not
Date_isPublicHoliday( aDate, "" ) ) )
end function

' *
' * @param  aFromDate       Beginn der Zeitperiode
' * @param  aToDate         Ende der Zeitperiode
' * @return Anzahl der Arbeitstage in der angegebenen Zeit
' * @author Joerg Jahnke (JJ)   19.11.97
' */
'    public double Date_getWorkdays( Date aFromDate, Date aToDate ) {}
function Date_getWorkdays( aFromDate, aToDate )
    const BASEDATE  = "1/1/1900"
    Dim i
    Dim nDays, nFromDate, nToDate

    nDays = 0
    nFromDate = Date_dayDiff( BASEDATE, aFromDate )
    nToDate = Date_dayDiff( BASEDATE, aToDate )

    for i = nFromDate to nToDate
        Dim aDate

        aDate = DateAdd( "d", i, BASEDATE )

        if Date_isWorkDay( aDate ) then
            nDays = nDays + 1
            if Date_isHalfPublicHoliday( aDate, "" ) then nDays = nDays - 0.5
        end if
    next

    Date_getWorkdays = nDays
end function


'/**
' * convert date between formats.
' *
' * @param  sDate          string to be converted
' * @param  aSrcFormat     source format
' * @param  aTgtFormat     target format
' * @return string in changed format
' */
'    public String Date_convertFormat( String sDate, String aSrcFormat, String
aTgtFormat ) {}
function Date_convertFormat( sDate, aSrcFormat, aTgtFormat )
    Dim aDateStr

    aDateStr = "" & sDate
    if aDateStr <> "" and not IsNull( aDateStr ) then

        Dim  nIndex,  aSrcSeparator,  aSrcArray,  aTgtSeparator,  aTgtArray,
aDateSrcArray, nBound, sTime

        sTime = ""

        if UBound( Split( aDateStr ) ) >= 1 then
            sTime = Split( aDateStr )( 1 )
            aDateStr = Split( aDateStr )( 0 )
        end if

        nIndex = InStr( aSrcFormat, "." )
        if nIndex > 0 then
            aSrcSeparator = "."
        else
            aSrcSeparator = "/"
        end if
        aSrcArray = Split( aSrcFormat, aSrcSeparator )
        aDateSrcArray = Split( aDateStr, aSrcSeparator )
        nBound = UBound( aDateSrcArray )

        nIndex = InStr( aTgtFormat, "." )
        if nIndex > 0 then
```

```
            aTgtSeparator = "."
        else
            aTgtSeparator = "/"
        end if
        aTgtArray = Split( aTgtFormat, aTgtSeparator )

        Dim aDateTgtArray()
        Dim aIntArray()
        Redim aDateTgtArray( nBound )
        Redim aIntArray( nBound )

        Dim i,j

        for j = 0 to nBound
            for i = 0 to nBound
                if left( aTgtArray( j ), 1 ) = left( aSrcArray( i ), 1 ) then
                    aIntArray( j ) = i
                    exit for
                end if
            next
        next

        for i = 0 to nBound
            aDateTgtArray( i ) = Right( "000" + aDateSrcArray( aIntArray( i ) ),
Len( aTgtArray( aIntArray( i ) ) ) )
        next

        Date_convertFormat = Join( aDateTgtArray, aTgtSeparator )
        if sTime <> "" then Date_convertFormat = Date_convertFormat + " " + sTime
    else
        Date_convertFormat = ""
    end if
end function


<%
 '<!-- #include virtual = "/Inc/121/International.inc" -->
%>


<%
'public class Array { private Array() {}

'/**
' * determine Index of an element in an array.
' *
' * @param  aArray          array to be searched
' * @param  aObject         object to be searched
' * @return -1, if object not found, else index of object in array

' */
'    public int Array_indexOf( Object[] aArray, Object aObject ) {}
function Array_indexOf( aArray, aObject )
    Dim aElement
    Dim i

    Array_indexOf = -1
    i = 0
    for each aElement in aArray
        if aElement = aObject then Array_indexOf = i : exit for
        i = i + 1
    next
end function

'/**
' * determine index of an element in array. comparison not case sensitive.
' *
' * @param  aArray          array to be searched
' * @param  aObject         object to be searched
' * @return -1, if object not found, else index of object in array

' */
'    public int Array_indexOf( Object[] aArray, Object aObject ) {}
function Array_indexOfUCase( aArray, aObject )
    Dim aElement
    Dim i

    Array_indexOfUCase = -1
    i = 0
    for each aElement in aArray
        if UCase( aElement ) = UCase( aObject ) then Array_indexOfUCase = i :
exit for
        i = i + 1
```

```
    next
end function

'/**
' * Test whether object present in array.
' *
' * @param  aSearchStr      object to be searched
' * @param  aArray          array to be searched
' * @return TRUE, if object present, else FALSE
' */
'    public boolean Array_contains( Object[] aArray, Object aSearchStr ) {}
function Array_contains( aArray, aSearchStr )
    Array_contains = ( Array_indexOf( aArray, aSearchStr ) >= 0 )
end function

'/**
' * Test whether character string present in array. comparison not case
sensitive.
' *
' * @param  aSearchStr      object to be searched
' * @param  aArray          array to be searched
' * @return TRUE, if object present, else FALSE
' */
'    public boolean Array_containsUCase( String[] aArray, String aSearchStr ) {}
function Array_containsUCase( aArray, aSearchStr )
    Array_containsUCase = ( Array_indexOfUCase( aArray, aSearchStr ) >= 0 )
end function


'/**
' * remove object from array' *
' * @param  aRemoveStr      object to be removes
' * @param  aArray          array to be searched
' * @return copy of original array without object to be removed
' */
'    public Object[] Array_removeElement( Object aRemoveStr, Object[] aArray ) {}
function Array_removeElement( aArray, aRemoveStr )
    Dim aResultArray()
    Dim i
    Dim aString
    Dim bComp

    i = 0
    for each aString in aArray
        bComp = ( VarType( aRemoveStr ) = VarType( aString ) )
        if bComp then
            if IsObject( aRemoveStr ) then
                bComp = ( aRemoveStr is aString )
            else
                bComp = ( aRemoveStr = aString )
            end if
        end if
        if not bComp then
            Redim preserve aResultArray( i )
            if IsObject( aString ) then
                Set aResultArray( i ) = aString
            else
                aResultArray( i ) = aString
            end if
            i = i + 1
        end if
    next

    Array_removeElement = aResultArray
end function

'/**
' * remove element with given index from array.
' *
' * @param  aArray          array to be processed
' * @param  nIndex      index of the element to be removed
' * @return copy of original array without object to be removed
' */
'    public Object[] Array_remove( Object[] aArray, int nIndex ) {}
function Array_remove( aArray, nIndex )
    Array_remove = aArray
    on error resume next
    Array_remove = Array_removeElement( aArray, aArray( nIndex ) )
end function
```

```
'/**
' * create copy of array. (Shallow-copy)
' *
' * @param  aArray      array to be copied
' * @return copy of array
' */
'    public Object[] Array_copy( Object[] aArray ) {}
function Array_copy( aArray )
    Dim aResultArray()
    Dim n, i

    n = Array_safeUBound( aArray )
    Redim aResultArray( n )

    for i = 0 to n
        if isObject( aArray( i ) ) then
            Set aResultArray( i ) = aArray( i )
        else
            aResultArray( i ) = aArray( i )
        end if
    next

    Array_copy = aResultArray
end function

'/**
' * append object to end of array.
' *
' * @param  aNewStr          object to be appended
' * @param  aArray           array to be processed
' * @return copy of original array with the object at the end of the array
' */
'    public Object[] Array_append( Object[] aArray, Object aNewStr ) {}
function Array_append( aArray, aNewStr )
    Dim aResultArray
    Dim n

    aResultArray = Array_copy( aArray )
    ' Array um ein Element verlaengern
    n = Array_safeUBound( aResultArray ) + 1
    Redim preserve aResultArray( n )

    if isObject( aNewStr ) then
        Set aResultArray( n ) = aNewStr
    else
        aResultArray( n ) = aNewStr
    end if

    Array_append = aResultArray
end function

'/**
' * array for amending content of a further array.
' *
' * @param  aArray          array to be processed
' * @param  aAppendArray    array with new contents
' * @return  original array with the contents of the second array at the end of
      the array
' */
'    public void Array_append2( Object[] aArray, Object[] aAppendArray ) {}
sub Array_append2( aArray, aAppendArray )
    Dim nAppendSize

    nAppendSize = Array_safeUBound( aAppendArray )
    if nAppendSize >= 0 then
        Dim nOldSize, nNewSize
        Dim aElement
        Dim i

        nOldSize = Array_safeUBound( aArray )
        nNewSize = nOldSize + nAppendSize + 1

        Redim preserve aArray( nNewSize )
        i = nOldSize + 1
        ' alle Elemente kopieren
        for each aElement in aAppendArray
            if isObject( aElement ) then
                Set aArray( i ) = aElement
            else
                aArray( i ) = aElement
            end if
            i = i + 1
```

```
        next
    end if
end sub

'/**
' * determine upper limit of array.
' *
' * @param  aArray      array to be processed
' * @return integer of upper limit,
' */
'    public int Array_safeUBound( Object aArray ) {}
function Array_safeUBound( aArray )
    Array_safeUBound = -1
    on error resume next
    Array_safeUBound = UBound( aArray )
end function

'/**
' * access list element with given index.
' *
' * @param  aArray      list
' * @param  nIndex      index of element
' */
'    public Object Array_safeGet( Object[] aArray, int nIndex ) {}
function Array_safeGet( aArray, nIndex )
    Array_safeGet = null
    on error resume next
    Set Array_safeGet = aArray( nIndex )
    Array_safeGet = aArray( nIndex )
end function


'/**
' * sort entries of array' *
' * @param  aArray      array to be sorted
' * @return sorted array
' */
'    public Object[] Array_sort( Object[] aArray ) {}
function Array_sort( aArray )
    Dim aResultArray

    aResultArray = Array_copy( aArray )

    Dim bFinish
    Dim i, n
    Dim aElement

    n = Array_safeUBound( aResultArray )
    bFinish = false
    while not bFinish
        bFinish = true

        for i = 0 to n - 1
            ' Falsche Reihenfolge ?
            if aResultArray( i ) > aResultArray( i + 1 ) then
                ' Dann vertauschen
                aElement = aResultArray( i )
                aResultArray( i ) = aResultArray( i + 1 )
                aResultArray( i + 1 ) = aElement
                ' Ein weiterer Durchlauf muss folgen
                bFinish = false
            end if
        next
    wend

    Array_sort = aResultArray
end function
'}
%>
```

Claims

1. Method of providing resources adapted to a user environment, including

   - obtaining a user parameter for setting the user environment,

   - executing an application independent of the user environment including reading a resource identifier independent of the user parameter,

   - loading a lookup object linking the resource identifier and resource data dependent on the user parameter, and

   - calling a lookup function for obtaining from the lookup object resource data based on the read resource identifier.

2. Method of claim 1, wherein the resource data are constituted by at least on one of the group consisting of

   - an array of at least one character, and

   - a resource function including rules for character representation.

3. Method of claim 2, wherein the resource function includes rules for at least one of the group consisting of

   - date representation,

   - time representation,

- currency representation, and

- floating point representation.

4. Method of one of the claims 1 - 3, wherein the user environment includes at least one of the group consisting of

- a location, and

- a user language preference.

5. Method of one of the claims 1 - 4, wherein for each user at least one lookup object is loaded.

6. Method of one of the claims 1 - 4, wherein for a plurality of users at least one lookup object is loaded.

7. Method of one of the claims 1 - 6, wherein

- the lookup function includes

  - generating a string identifier consisting of the resource identifier and the user parameter,

  - performing the lookup function using the string identifier, and

- the lookup object includes an assignment between the string identifier and the resource data dependent on the user parameter.

8. Method of one of the claims 1 - 6, wherein the lookup function includes calling a dictionary function for obtaining one of a plurality of lookup objects

corresponding to the user parameter and which links the resource identifier with the resource data, and

9. Method of one of the preceding claims 1 - 6, wherein the lookup object links the resource identifier with a plurality of user parameters and each of the plurality of user parameters with resource data.

10. Method of one of the preceding claims, wherein

- the application is executed at an application means and is controlled by a client,

- the user parameter is received at the application means from the client, and

- the resource data are transmitted to the client.

11. Method of one of the preceding claims, further including

- reading resource data, and

- calling the lookup function for obtaining from the lookup object a resource identifier based on the read resource data.

12. Method of one of the preceding claims, wherein the user parameter is stored in an object.

13. Method of one of the preceding claims, wherein the user parameter is passed from a client to an application server executing the application.

14. Method of one of the preceding claims, wherein a programming language is used for implementing the method of one of the claims 1 - 13.

15. Method of one of the preceding claims, wherein the programming language is a scripting language.

16. Method of one of the preceding claims, wherein said scripting language is the Visual Basic Scripting language.

17. Method of one of the preceding claims, wherein said programming language is the Visual Basic Scripting language and said program is in the form of Microsoft Active Server Pages.

18. A computer program provided in accordance with the method of one of the preceding claims.

19. A computer readable medium, having recorded thereon the program according to claim 17.

20. A computer program product comprising the computer readable medium according to claim 18.

21. Resource means for providing resources depending on a user environment, including

- user parameter means (11) for obtaining a user parameter for setting the user environment,

- application means (12) for executing an application independent of the user parameter including reading a resource identifier independent of the user parameter, and

- lookup means (13) for initializing a lookup object linking the resource identifier and resource data, the resource data being dependent on the user parameter, and for calling a lookup function for

obtaining from the lookup object resource data based on the read resource identifier.

22. Resource means of claim 21, wherein the resource data are constituted by at least on one of the group consisting of

- an array of at least one character, and
- a resource function including rules for character representation.

23. Resource means of one of the claims 21 and 22, wherein the resource function includes rules for at least one of the group consisting of

- date representation,
- time representation,
- currency representation, and
- floating point representation.

24. Resource means of one of the claims 21 - 23, wherein the user environment includes at least one of the group consisting of

- a location, and
- a user language preference.

25. Resource means of one of the claims 21 - 24, including means for loading at least one lookup object for each user.

26. Resource means of one of the claims 21 - 24, including means for loading at least one lookup object for a plurality of users.

27. Resource means of one of the claims 21 - 26, wherein

- the lookup means includes
  - means for generating a string identifier consisting of the resource identifier and the user parameter,
  - means for performing the lookup function using the string identifier, and
- the lookup object includes an assignment between the string identifier and the resource data dependent on the user parameter.

28. Resource means of one of the claims 21 - 26, comprising means for calling a dictionary function for obtaining one of a plurality of lookup objects corresponding to the user parameter and which links the resource identifier with the resource data.

29. Resource means of one of the claims 21 - 26, wherein the lookup object links the resource identifier with a plurality of user parameters and each of the plurality of user parameters with resource data.

30. Resource means of one of the claims 21 - 29, wherein

- the application executed at the application means is controlled by a client,
- the user parameter means is arranged for receiving the user parameter from the client, and

- the resource means is arranged to transmit the resource data to the client.

31. Resource means of one of the claims 21 - 30, wherein

- the application means is adapted to read resource data, and

- the lookup means is adapted to obtain from the lookup object a resource identifier based on the read resource data.

32. Resource means of one of the claims 21 - 31, wherein the user parameter is stored in an object.

33. Resource means of one of the claims 21 - 32, wherein the user parameter is passed from a client to an application server executing the application.

34. Client adapted to cooperate with the resource means of one of the claims 21 - 33, including

- means for setting the user parameter, and

- means for transmitting the user parameter to the data processing device.

35. Data processing system for providing resources depending on a user environment, comprising a client and resource means, wherein

- the client includes

  - user parameter means for setting a user parameter for selecting the user environment, and

- means for transmitting the user parameter to the data processing device,

- the resource means includes:

  - user parameter means for obtaining a user parameter for setting the user environment,

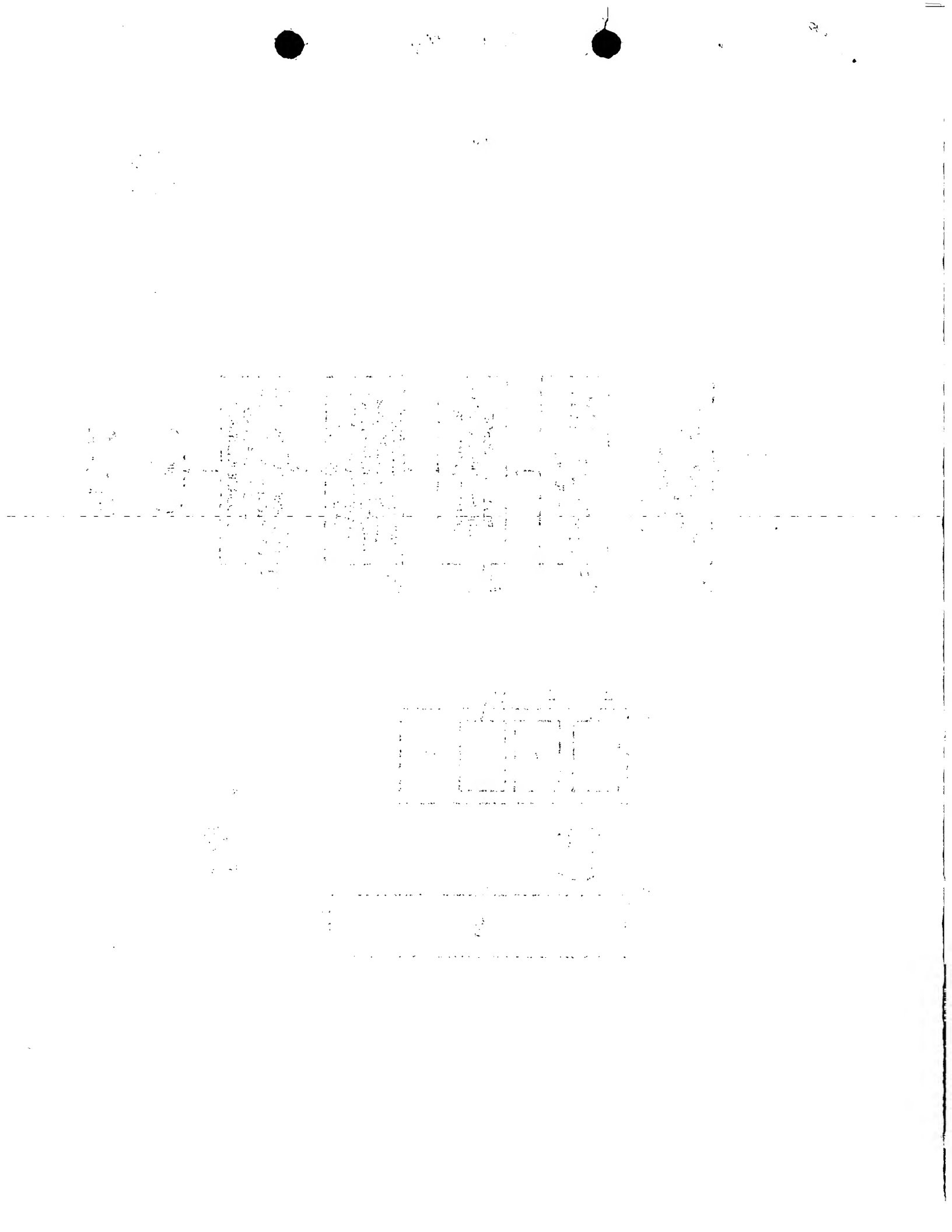
  - application means for executing an application independent of the user environment including reading a resource identifier independent of the user parameter, and

  - lookup means for calling a lookup function for obtaining from the lookup object resource data based on the read resource identifier and for loading a lookup object linking the resource identifier and resource data dependent on the user parameter.

Abstract

Method and apparatus for providing resources adapted to a user environment wherein user parameter means obtain a user parameter for selecting a user environment, for example a location or a user language preference upon user input. Application means execute an application independent of the user parameter including reading a resource identifier independent of the user parameter, e.g. during generating a web page. Lookup means load a lookup object which links the resource identifier and resource data, the resource data being dependent on the user parameter, e.g. data to be presented to a user in a particular language and/or format. The lookup means calls a lookup function for obtaining from the lookup object resource data based on the read resource identifier. Thus, applications may be written independent of a user environment, i.e. of the selected user parameter, and information may be presented to a user in a local format or language comfortable for the user.

Fig. 1



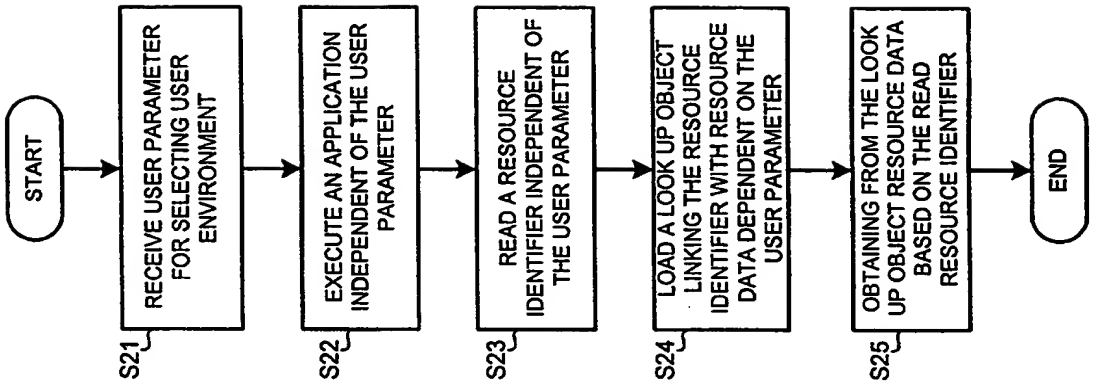


Fig. 2

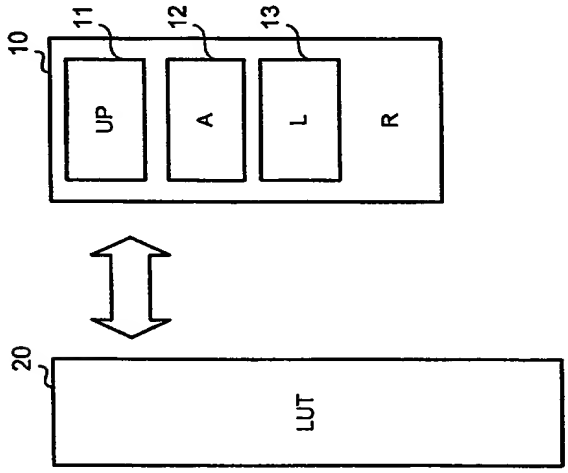


Fig. 1

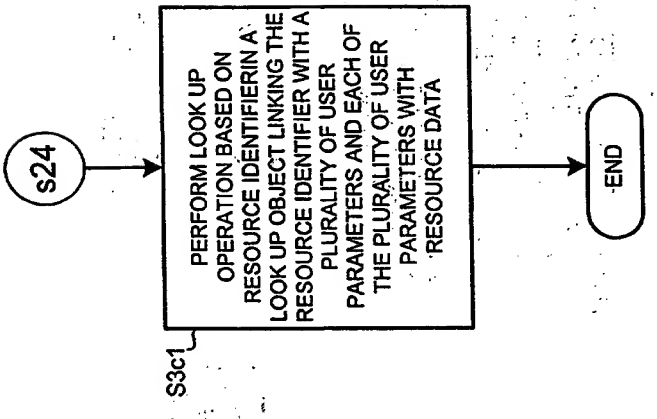


Fig. 3a

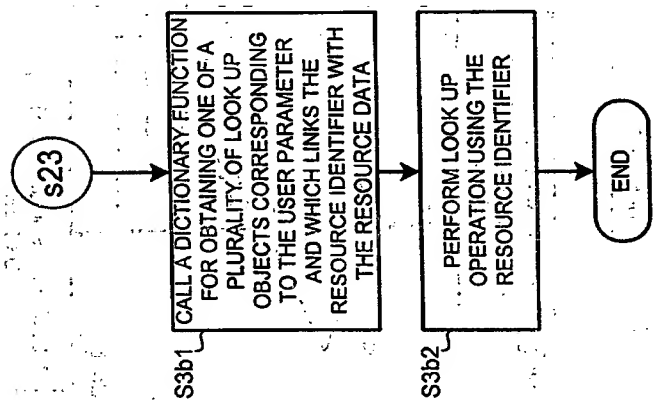


Fig. 3b

Fig. 3c

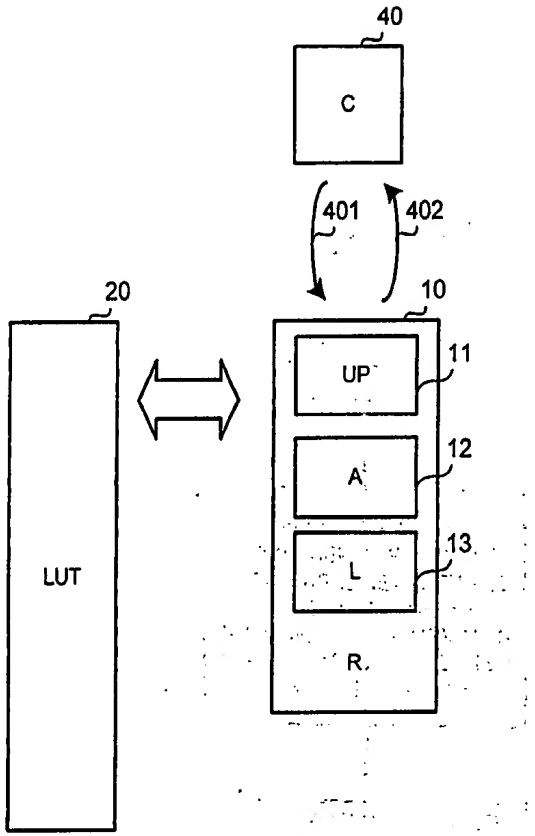


Fig. 4

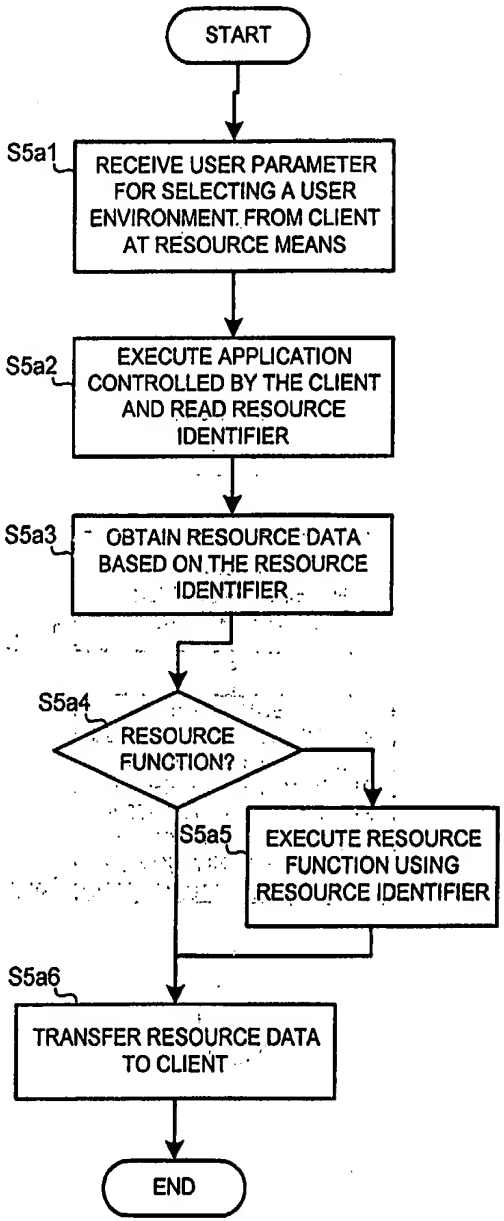


Fig. 5a

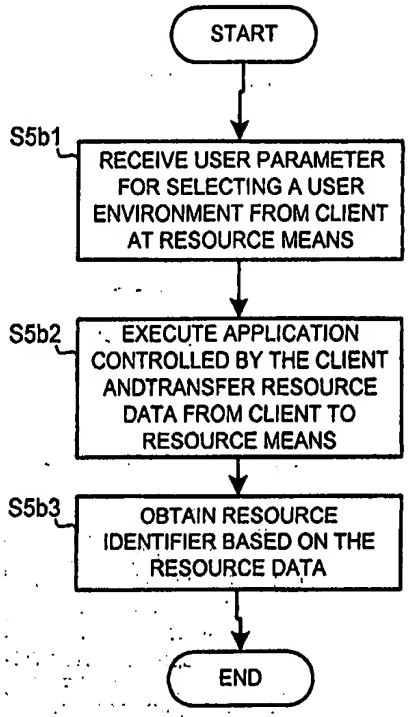


Fig. 5b

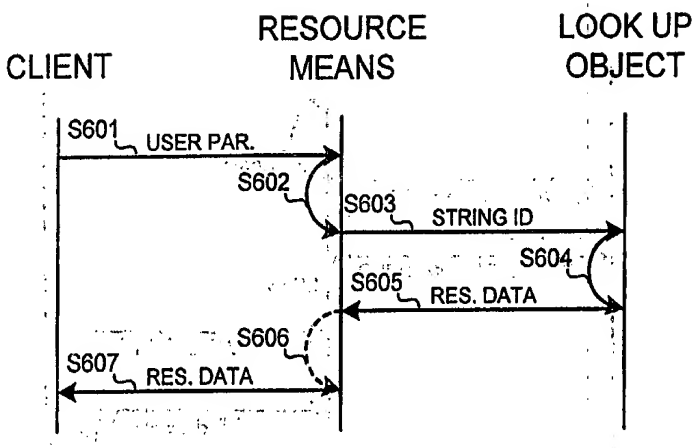


Fig. 6

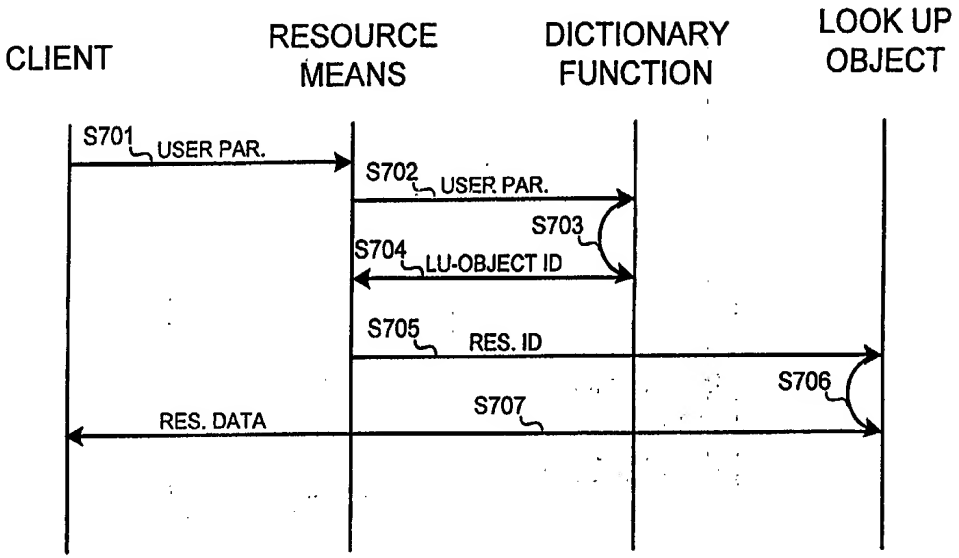


Fig. 7

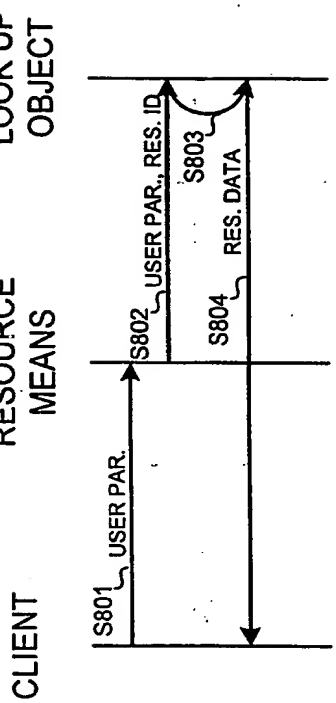

CLIENT
RESOURCE MEANS
LOOK UP OBJECT
S801
USER PAR.
S802
USER PAR., RES. ID
S803
S804
RES. DATA

Fig. 8

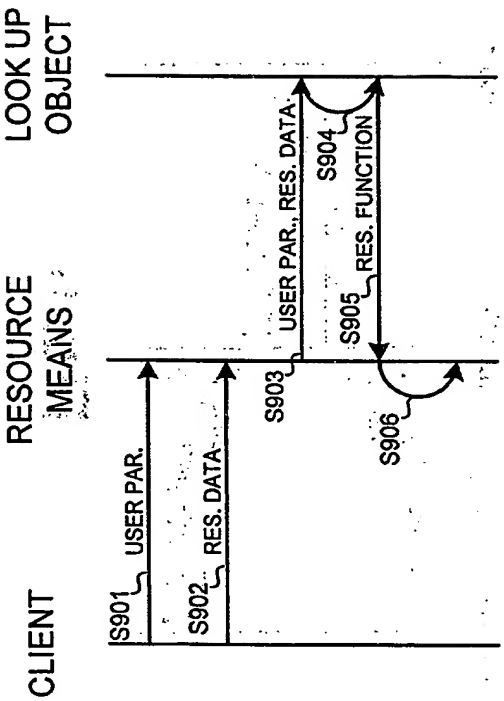

CLIENT
RESOURCE MEANS
LOOK UP OBJECT
S901
USER PAR.
S902
RES. DATA
S903
USER PAR., RES. DATA
S904
S905
RES. FUNCTION
S906

Fig. 9